Annual Report

Grant No. NAG-1-349

DIGITAL CONTROL SYSTEM FOR SPACE STRUCTURE DAMPERS

Submitted to:

National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23665

Attention: Dr. Garnett C. Horner
SDD, MS 230

Submitted by:

J. K. Haviland
Professor

Department of Mechanical and Aerospace Engineering
SCHOOL OF ENGINEERING AND APPLIED SCIENCE
UNIVERSITY OF VIRGINIA
CHARLOTTESVILLE, VIRGINIA 22901

(NASA-CR-181253) DIGITAL CCNTECL SYSTEM FOR SPACE STRUCTUBE DAMPEBS Annual Beport (Virginia Univ.) 96 p Avail: NTIS HC A02/MF A01 CSCL 22B

N87-27704

Unclas
G3/18 0093175

Report No. UVA/528224/MAE86/105

September 1985

Copy No.___

ABSTRACT

This is the final report of a two-year study of digital control systems for space structural dampers, or more specifically, for proof-mass dampers or actuators. Previously, a proof-mass actuator had been developed, of which twelve had been delivered to NASA, and analog and digital control systems had been developed in prototype form. Under the first year of the present study, a Z80 controller was developed, slaved to a TRS80 microcomputer. During the final year, which is covered in this report, a digital controller was developed using an SDK-51 System Design Kit, which incorporates an 8031 microcontroller. As part of this study, the necessary interfaces were installed in the wire-wrap area of the SDK-51 and a pulse-width modulator was developed to drive the coil of the actuator. Also, control equations were developed, using floating- point arithmetic. The design of the digital control system is emphasized in this report, and it is shown that, provided certain rules are followed, an adequate design can be achieved. It is recommended that the so-called w-plane design method be used, and that the time elapsed before output of the up-dated coil-force signal be kept as small as possible. However, the cycle time for the controller should be watched carefully, because very small values for this time can lead to digital noise.

ACKNOWLEDGEMENT

The gift by the INTEL Corporation of an SDK-51 System Design Kit is gratefully acknowledged. Without this gift, much of the work reported here could not have been attempted.

TABLE OF CONTENTS

LIST OF FIGURES

DEFINITIONS

a_0, a_1 = Coefficients of polynomial

a_F,A_F = Structural acceleration

b_1 = Coefficient of polynomial

c = Design damping (Ns/m)
D(s), etc. = Transfer function
F = Coil force
g = Acceleration of gravity (9.81 m/s^2)
G = Analog gain
G* = Digital gain
H(s), etc. = Transfer function
$H_c(s)$, etc. = Complex damping

I_n = Integer form of n

k = Integer time-interval variable
k_A, etc. = Digital gain terms

k_{max} = Maximum synthetic stiffness

k_s = Synthetic stiffness (N/m)

K = Analog gains used in calibration of system
m = Integer time-count for data output
M = Proof-mass (kg)
n = Integer cycle-time count for calculation cycle
$R_c(s)$ = Response amplitude ratio

s = Laplace variable
t = Time variable
T = Calculation cycle time
T_0 = Basic time interval (256 microseconds)

u,U = control state or output variable
w = Transform variable
x,X = Input variable
z = Transform variable
Z{} = z-transform equivalent of a Laplace transform
ϕ_M = Phase margin

γ = Lag to lead frequency ratio
ζ = Accelerometer gain parameter
ν = w-Plane frequency
ω = s-Plane frequency

<u>Subscripts</u>:

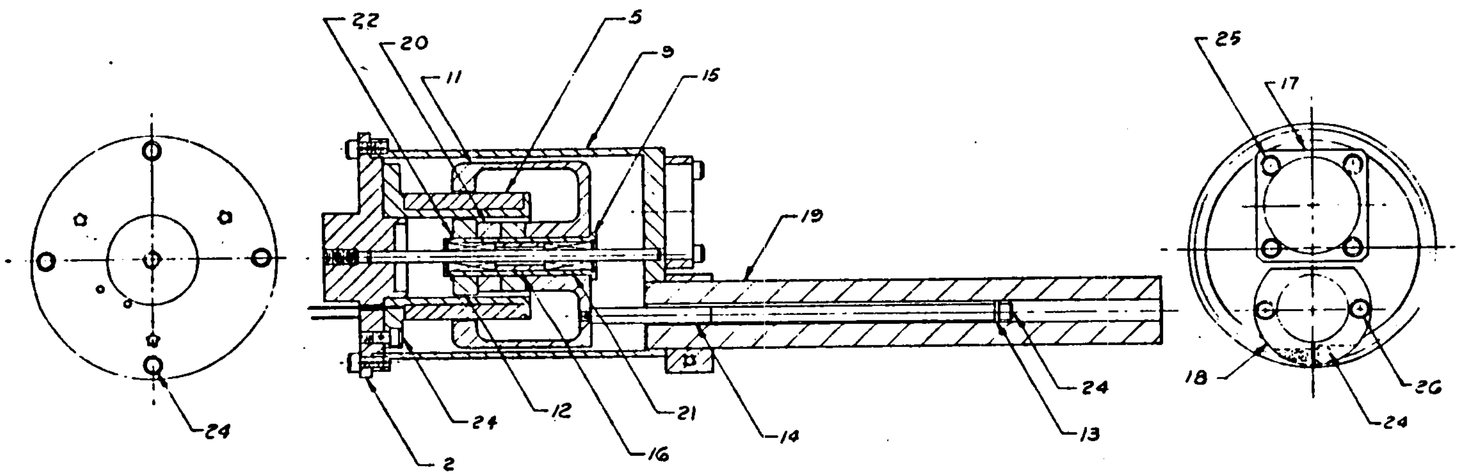
A = Accelerometer
B = Component of damping equation
c = Relating to damping
C = Coil

D = Relative proof-mass motion
L = LVDT
P = Proximeter
s = Relating to stiffness
V = Component of synthetic stiffness equation

INTRODUCTION

Discussion: This report covers the second year of a study of space structure damping under NASA Grant No. NAG-1-349, following Proposal No. MAE-NASA-2548-83 (1). Earlier, a general study of possible damper configurations had been reported under NASA Grant No. NAG-1-137-1 (2). Following that work, purchase order No. L46164B had been received from NASA for the design and construction of twelve proof-mass actuators, also referred to as space structure dampers. A sectioned assembly drawing for this design is shown as Figure 1. During these last two years, Mr. Michael Mallette, a doctoral candidate, has worked on the development of control laws under a NASA student fellowship. His dissertation is imminent. Under the present two-year grant, earlier reports (3,4) have covered design of the proof-mass actuator, and development of analog and Z80 controllers. The work reported here covers development of an 8051 series controller exclusively.

Equipment: The work on the 8051 series controllers was aided considerably by the donation of an SDK-51 System Design Kit from the INTEL Corporation. Also, two of the twelve NASA owned proof-mass actuators were obtained on loan, and were modified to take Bentley-Nevada Model 190 proximeter probes. This required two new cases, and tapered sleeves on the proof-masses, so that their position could be determined by proximeters. One of these actuators has been used by Mr. Mallette, this is shown in Figure 2 with an accelerometer which is also on loan from NASA. The other was used in the present study. It has a Sunstrand Model QA-900 accelerometer, a Bentley-Nevada 3106-2800-190 amplifier, and a



FOR ITEM DESCRIPTION SEE PARTS LIST

ACTIVE DAMPER
ASSEMBLY
12·28·82
J.H.D.

Figure 1. PROOF-MASS ACTUATOR SECTION

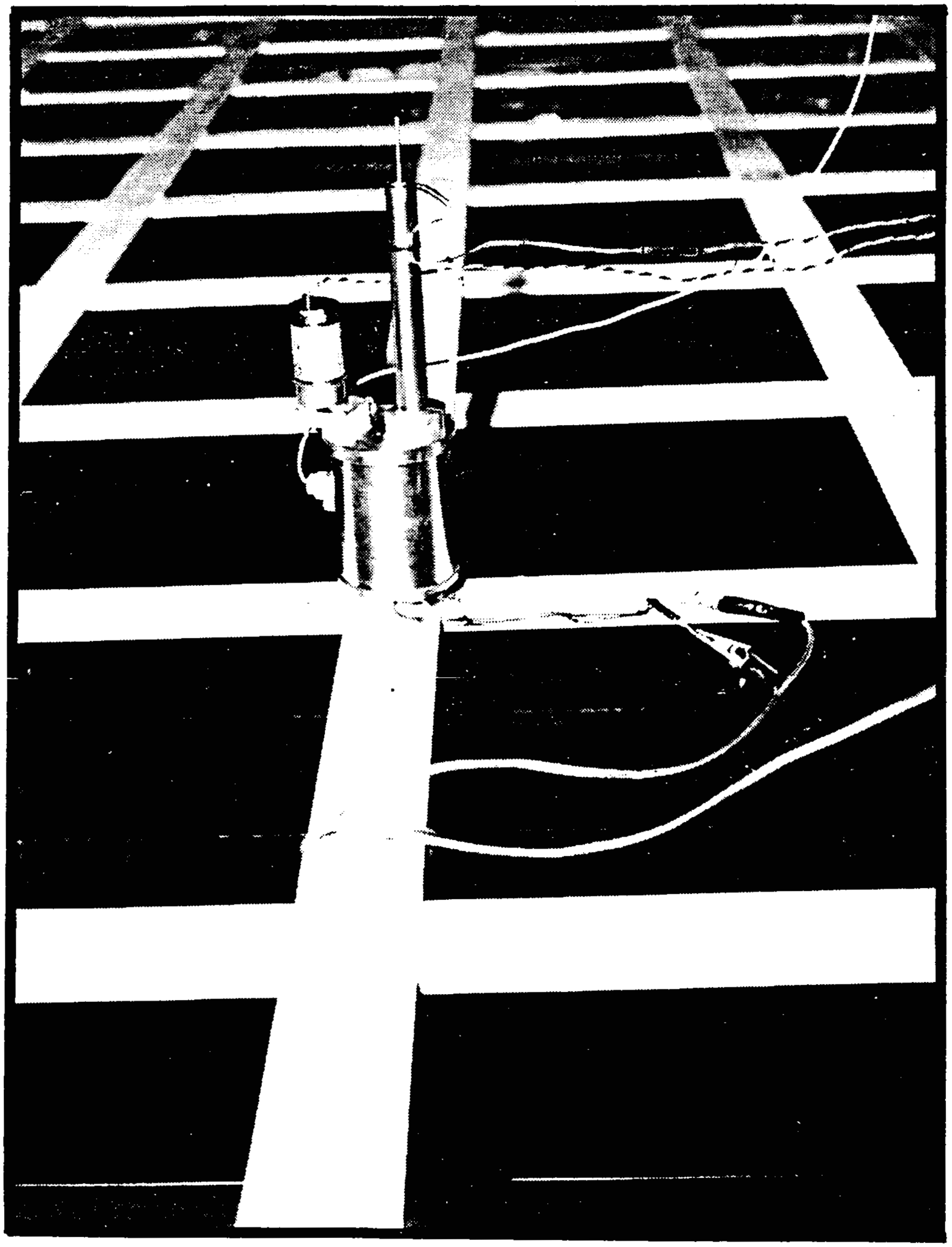


Figure 2. PROOF-MASS ACTUATOR WITH PROXIMETER

home-made pulse-width modulator (PWM) attached. A control system was built in the wire-wrap area of the SDK-51 board, as described later.

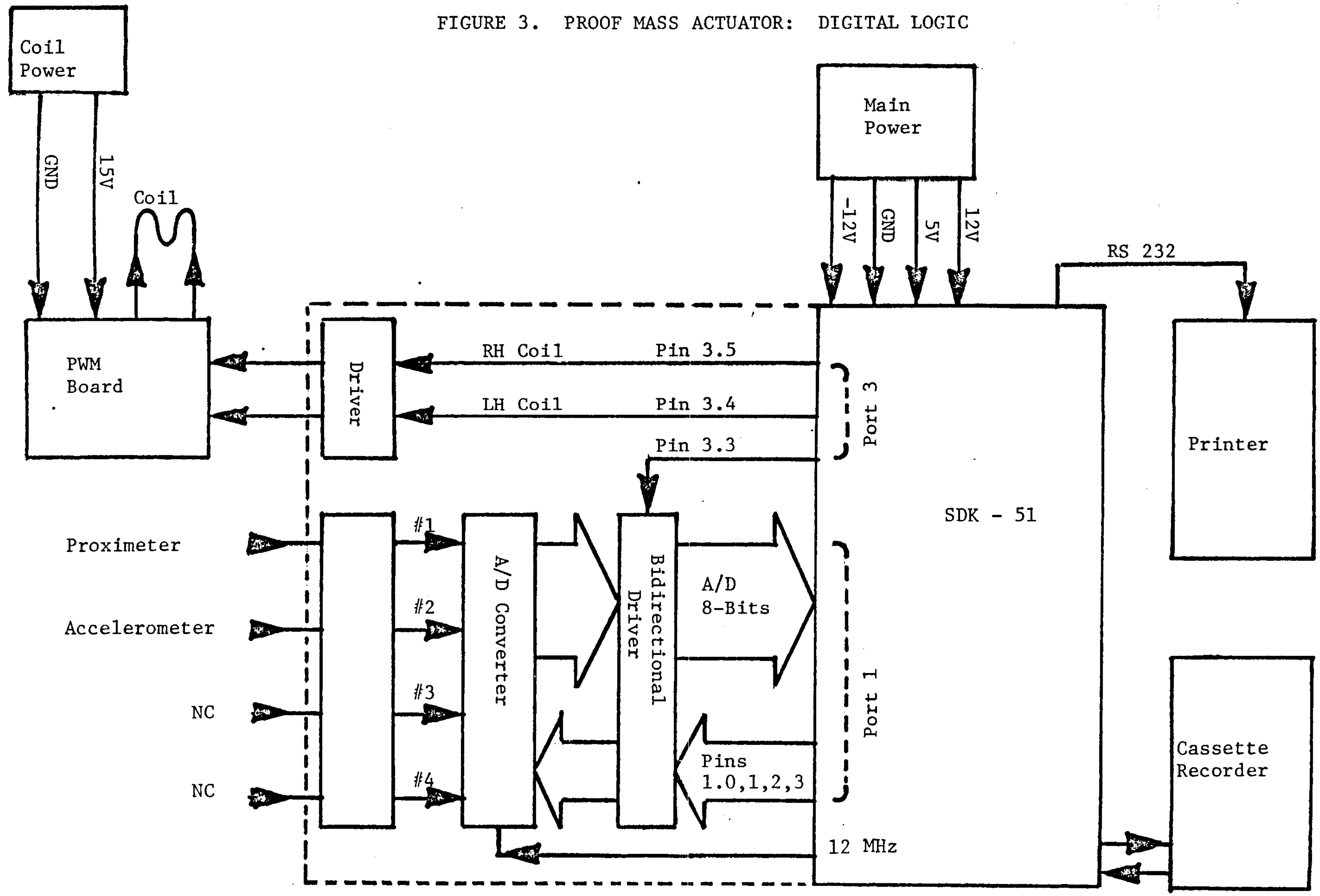
Work on the 8051 series: Work on the 8051 series controllers, which is literally an 8031, which has no internal program memory, was limited mainly to development of the system described above, and to the requisite SDK-51 programs, including two versions of the P1-D control realization first discussed in Reference 4. Behavior of the system was largely checked by simple observation, relying on Mr. Mallette's experience for further insight into its behavior. The following report covers a description of the controller hardware which was developed, and of the control program, together with computer predictions of the real damping vs. frequency, and of the relative amplitude of motion of the proof-mass within its case. The long general purpose SDK-51 program which was used is listed in Appendix A.

SDK-51 DEVELOPMENT BOARD

Description

An SDK-51 Development Board was obtained as a gift from the INTEL Corporation. Although it is designed for teaching the 8051 language, a wire-wrap area is provided for user experiments. This area was used to configure a controller for the proof-mass damper. Components in this area include four analog input ports (two populated), an A/D converter, and pulse-width modulated (PWM) outputs. An overall schematic of this system is shown in Figure 3,

FIGURE 3. PROOF MASS ACTUATOR: DIGITAL LOGIC



and logic diagrams are given in Appendix B.

As presently configured, 12 pins on two ports of the 8031 are used. These are all eight pins of port 1, and pins 3.3,4, and 5 of port 3. In addition, pin 3.0 has been programmed temporarily to indicated completion of digital calculations as an oscilloscope signal, but this could easily be discontinued. Pins 1.0,1,2, and 3 are connected to a transceiver, and can be used for output, otherwise, port 1 is used to read the A/D. Pin assignments are as follows:

Port 1	(Input)	Read A/D
Pin 1.0	(Output)	Input channel selection
Pin 1.1	"	" " "
Pin 1.2	"	Trigger A/D
Pin 1.3	"	Enable A/D
Pin 3.0	(Output)	Temporary oscilloscope signal.
Pin 3.3	"	Sets transceiver to output when high.
Pin 3.4	"	Sets L.H. end of coil to high voltage.
Pin 3.4	"	Sets R.H. end of coil to high voltage.

Four analog inputs were originally designed, but two will not be populated (LVDT and signal generator) until requirements for a slaved 8031 have been determined. The two which have been populated are #0, proximeter, and #1, accelerometer. The four inputs are selected by the outputs of Pins 1.0, and 1, through half of a 74LS139 decoder, and an LF13332 analog switch, with a TL087 high speed operational amplifier to improve output impedance. The selected signal is converted directly to 2's complement eight-bit form using a DAC0800 D/A and a DM2502 successive approximation register, with a LM361 high speed comparator to compare the two signals. Timing comes from the 12MHz crystal on the SDK-51 board, divided by powers of two in a 74LS163, as selected by jumpers. Signals are synchronized by a dual D-flip-flop, in a one-and-one-only configuration. A 75451 driver is used for the PWM output; the actual PWM function is carried out on a separate board attached to the proof-mass damper. This board consists of two pairs of Darlington transistors (NTE261 and NTE262), one pair is attached to each end of the coil, their bases are driven by 2N3904 transistors, which are themselves driven by 4N28 opto-electrical transistors from the PWM signals. With this arrangement, about +1 to -1 Amperes can be produced in the 8.5 Ohm coil. However, an important feature of this arrangement is that there is no coil current when both PWM signals are equal. Thus the coil does not heat up when the proof-mass damper is quiescent.

Comparison of 8051 with Z80:

The work reported here, in conjunction with the work reported for the previous year in Reference 4, affords an opportunity to compare the 8051 with the Z80, in the following ways:

Advantages of 8051 Series:

Multiplication: Only available on the 8051 series.

Division: Only available on the 8051 series, but of dubious value because it only produces the integer part of the quotient.

On-Board Timer: There are two onboard timers on the 8051 series, both with interrupts, whereas the same functions have to be provided by hardware with the Z80 (the 8052 series has an additional timer).

Interrupt Priority: There are two levels of interrupt priority, with a total of five interrupts (two timer, two general, and one serial). Again, this arrangement must be provided by hardware for the Z80.

Internal RAM: Internal RAM is provided on the 8051 series, with one page of byte addresses, plus another page of bit addresses covering part of the same field. One half page is devoted in each case to special function registers. This provides computing power unique to the 8051 series.

Internal UART: An internal UART on the 8051 series makes master-slave arrangements relatively simple. A third timer can be used to provide the needed Baud rate, or very high speed serial data exchange can be obtained using the clock-timer. In the master-slave arrangement, several slaves can be addressed individually.

Advantages of Z80:

16-Bit Arithmetic: Many operations can be carried out with 16 bits, compared to only 8 bits on the 8051 series.

BUSREQ: This feature of the Z80 permits a single slave arrangement in which the memory space of the slave is relatively easy to address. This proved to be a great advantage in the development of the Z80 system.

Vectored Interrupts: The Z80 can receive address vectors for interrupts, which simplifies the selection of different programs when running as a slave.

IN/OUT: The separate mapping of in/out memory space was an advantage, because these instructions could be decoded, and could be used to trigger operations such as read A/D. The same functions are obtained on the 8051 series by SETB and CLR instructions to the port pins.

The Z80 is Used in Small Computers: The fact that the Z80 is a well- known and popular computer chip was to its advantage in last year's work because it was relatively simple to use the Radio Shack Model 1 computer as a development system. A comparable system for the 8051 series, although considerably better, costs about ten times as much.

DERIVATION OF DIGITAL CONTROL EQUATIONS

Floating Point Subroutines

Since the INTEL 8051 series controller can only execute eight bit arithmetic, unlike the Z80 which can handle many sixteen bit operations, an early decision was made to use a sixteen (16) bit floating point format, with a signed seven bit mantissa and exponent, as follows:

Bit #	15	14 13 12 11 10 9 8	7	6 5 4 3 2 1 0
	sign of	mantissa	sign of	exponent

mantissa exponent

Thus +1 becomes .40X01, and -1 becomes .80X00, where X stands for exponent, and the decimal point means that the mantissa is fractional. We cannot use E for the exponent, as with decimals, because it is a hexadecimal digit.

The following subroutines are available:

ADD	NEGATE
SUBTRACT	MOVE IN MEMORY
MULTIPLY	FIXED TO FLOATING- FROM MEM.
STORE ABOVE RESULTS	FIXED TO FLOATING- FROM ACCUM.
IN MEMORY	FLOATING TO FIXED

These subroutines are identified in the listing supplied in the Appendix. Results of all operations except FLOATING TO FIXED are normalized by shifting ones into positive numbers and zeros into negative numbers, thus:

.00X00	becomes	.7FXF9
.3FX00	"	.7FXFF
.FFX00	"	.80XF9
.C0X00	"	.80XFF

Sometimes, the application of an operation and its inverse, such as ADD and SUBTRACT, or NEGATE NEGATE, results in a change in the last bit. Also, if exponent overflow occurs, it is replaced by X7F or X80, as appropriate, but the mantissa is meaningless. Further, the difference of two equal numbers leaves a zero mantissa, which is then normalized as a positive number. Thus the final result has a mantissa of .7F, while the exponent is reduced by 7.

The program in Appendix A includes a floating point calculator simulation program, similar to the reverse Polish system on the Hewlett-Packard calculators. The program includes all of the subroutines listed, with the exception of FIXED TO FLOATING, which is covered by the NORMALIZE operation. In addition, numbers can be entered into the display on the SDK 51 board, and the command ENTER can then enter them into a three tier stack, while READ can bring them back into the display. Operations on two numbers involve the SDK 51 display and the first number on the stack, the result is displayed, and the stack is moved down by one. The calculator program was written to permit development of the floating point subroutines, and to make it easier to calculate parameters to be used in experimental programs. It includes provisions for inserting floating-point parameters into data memory for use in the control programs.

Often it is necessary to find the floating point equivalent of a decimal number for insertion into the controller program. The following procedure was found to be useful:

(a) Express in form $M \times 2^{E}$

(b) Convert to form $.MH \times 2^{(EH+7)}$

(c) Write in form .MHX(EH+7) for entry onto board.

(d) Write in form MH,(EH+7) for entering into memory.

Note: M,E are decimal integers, with M between 63 and 127, while MH and EH are hexadecimal equivalents to 2 places.

Example: Convert 0.0287

(a) $.0287 = 117 \times 2^{-12}$

(b) $= .75H \times 2^{-12+7} = .75H \times 2^{FBH}$

(c) $= .75XFB$

(d) $= 75H,FBH$

To find the decimal equivalent , this process is reversed:

(a) Express in form $MH \times 2^{EH-7}$

(b) Convert to form $M \times 2^{E-7}$

(c) Evaluate

Example: Convert .75XFB

(a) $.75XFB = 75H \times 2^{FBH-7} = 75H \times 2^{F4H}$

(b) $= (7 \times 16 + 5) \times 2^{-12} = 117 \times 2^{-12}$

(c) $= 117/4096 = 0.0286$

Digital Program by Rectangular Rule

The program shown in Appendix A is based on the rectangular rule of integration, but such refinements as zero-order-hold and computational delay have been omitted. The program corresponds very closely to the P1-D program of Reference 1, except that the

16-bit arithmetic of the Z80 has been replaced by the floating point arithmetic described in the preceding paragraph, and the divide by powers-of-two operations have been replaced by full multiplications.

The system to be investigated is shown in block diagram form in Figure 4. Some changes in notation have been made relative to Reference 4, mainly the replacement of number subscripts to avoid confusion with state-space notation, and a redefinition of H_A.

From Figure 4:

$$F(s) = H_A(s)A_F(s) - H_P(s)X_D(s)$$

while, from the dynamics of the proof mass

$$F(s) = MA_F(s) + Ms^2X_D(s)$$

The signal generator input, x_S, has not been included in these equations. We can now develop two functions which are of considerable importance in the evaluation of damper performance:

$$\begin{aligned} H_C(s) &= sF(s)/A_F(s) \\ &= s\{(H_A(s) + H_P(s)/s^2)/(1 + H_P(s)/Ms^2)\} \end{aligned}$$

$$\begin{aligned} R_C(s) &= s^2X_D(s)/A_F(s) \\ &= -\ (1 - H_A(s)/M)/(1 + H_P(s)/Ms^2) \end{aligned}$$

where H_C is the complex damping, whose real part must be positive at any frequency at which energy is to be absorbed, and R_C is the ratio of the proof-mass amplitude to that of the structure. For example, if its norm is 2, then the proof mass will just hit the

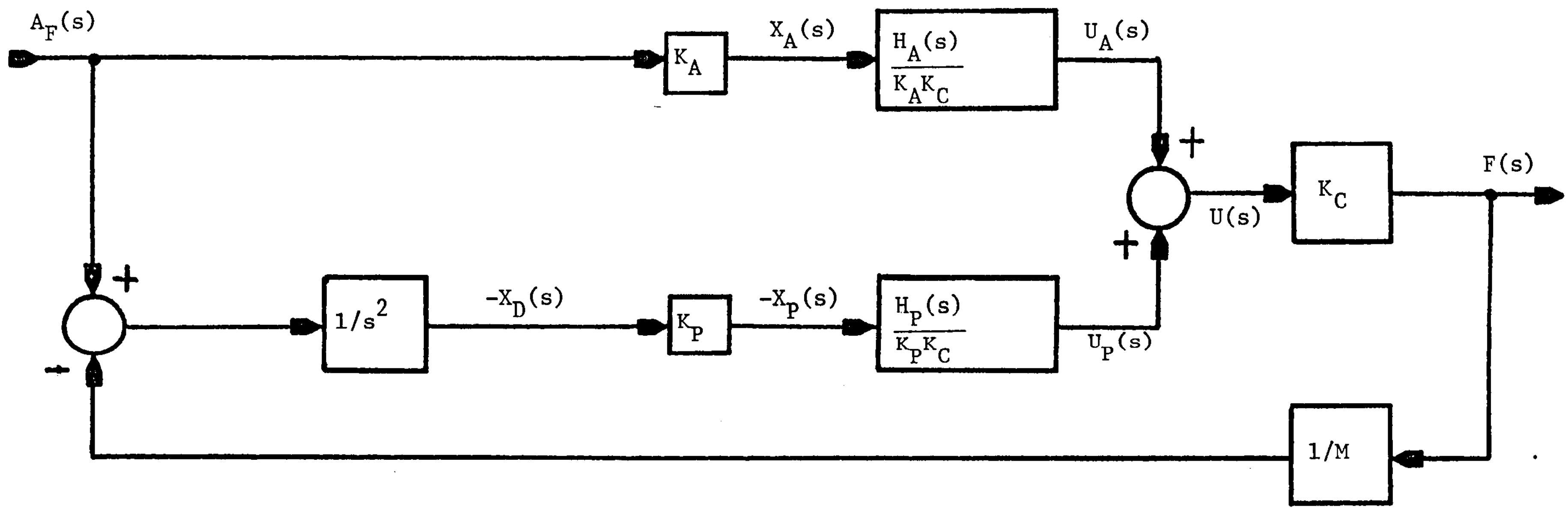


FIGURE 4. PROOF-MASS ACTUATOR CONTROLS: ANALOG

stops of a one inch stroke damper when the structural double-amplitude reaches a half inch.

It has been found that satisfactory values for Re{H_C} and Norm{H_C} can be obtained if the following rules are followed:

(1) There is a positive input to the A/D (this may mean a negative voltage, because most A/D's invert) when there is an acceleration directed from the structure to the damper, i.e., a positive acceleration.

(2) There is a positive input to the A/D when the proof-mass is against the structure, i.e., a negative displacement.

(3) A +/- 1g accelerometer range exactly covers the full input range to the A/D. (referred to as +/- 1 here, rather than to a range of voltages).

(4) The full range of proof-mass travel exactly covers the full input range to the A/D.

(5) The force exerted on the proof mass, when the accelerometer is attached, exactly balances its weight component.

(6) The synthetic spring stiffness, k_s, is a fraction of the maximum available value, k_{max}, chosen to give suitable centering behavior.

(7) At high frequency, H_A should approach the real value, c, of the required design damping.

(8) At high frequency, H_P should approach zero.

(9) The open loop gain, H_P/Ms^2, of the synthetic spring circuit should have an adequate phase margin.

Rules 1 and 2 ensure the correct polarity, and permit a simple evaluation of the damper using the DEMO modes described in the

Appendix. When this polarity is correct, the damper exhibits simple spring behavior or a tendency to remain centered when the damper assembly is tilted, according to which DEMO program is selected.

Rules 3 and 4 permit calibration of the system by one of the following methods:

(a) Direct monitoring of the A/D inputs with a voltmeter.

(b) Use of the DISPLAY subroutine described in the Appendix which displays the input in 2's complement hexadecimal form on the SDK-51 board.

(c) Use of the appropriate DEMO program together with monitoring of the output to the coil.

Applying these rules, we have:

$$K_A = 1/1g = 1/9.8 = 0.1020 \ s^2/m$$

$$K_P = (40 \text{ ins/m})/(1/2 \text{ inch amplitude}) = 80 \ m^{-1}$$

Rules 3 and 7 are satisfied if H_A has the form:

$$H_A(s) = 2M\zeta/(1+s/\omega_A)$$

with

$$\zeta = c/2M\omega_A$$

while rule 5 is satisfied when $\zeta=1/2$.

Rules 4,6 and 8 are satisfied if $H_P(s)$ has the form:

$$H_P(s) = k_s(1+s/\omega_V)/(1+s/\omega_P)$$

so that the open-loop transfer function is:

$$H_P(s)/Ms^2 = (\omega_N{}^2/s^2)(1+s/\omega)_V)/(1+s/\omega)_P)$$

where:

$$\omega_N{}^2 = k_s/M$$

and:

$$k_s < k_{max} = K_PK_C$$

From several measurements on the present damper design, when the current is adjusted to range from -1 to +1 Amps. over the full range of digital input:

$$K_C = 1.92 \text{ N}$$

thus:

$$k_{max} = K_PK_C$$

$$= (80 \text{ m}^{-1})(1.93 \text{ N}) = 155 \text{ N/m}$$

Rule 9 is satisfied if suitable values are picked for the two break frequencies in $H_P(s)$. Using the Bode plot of Figure 5, and designing for a phase margin of ϕ_M:

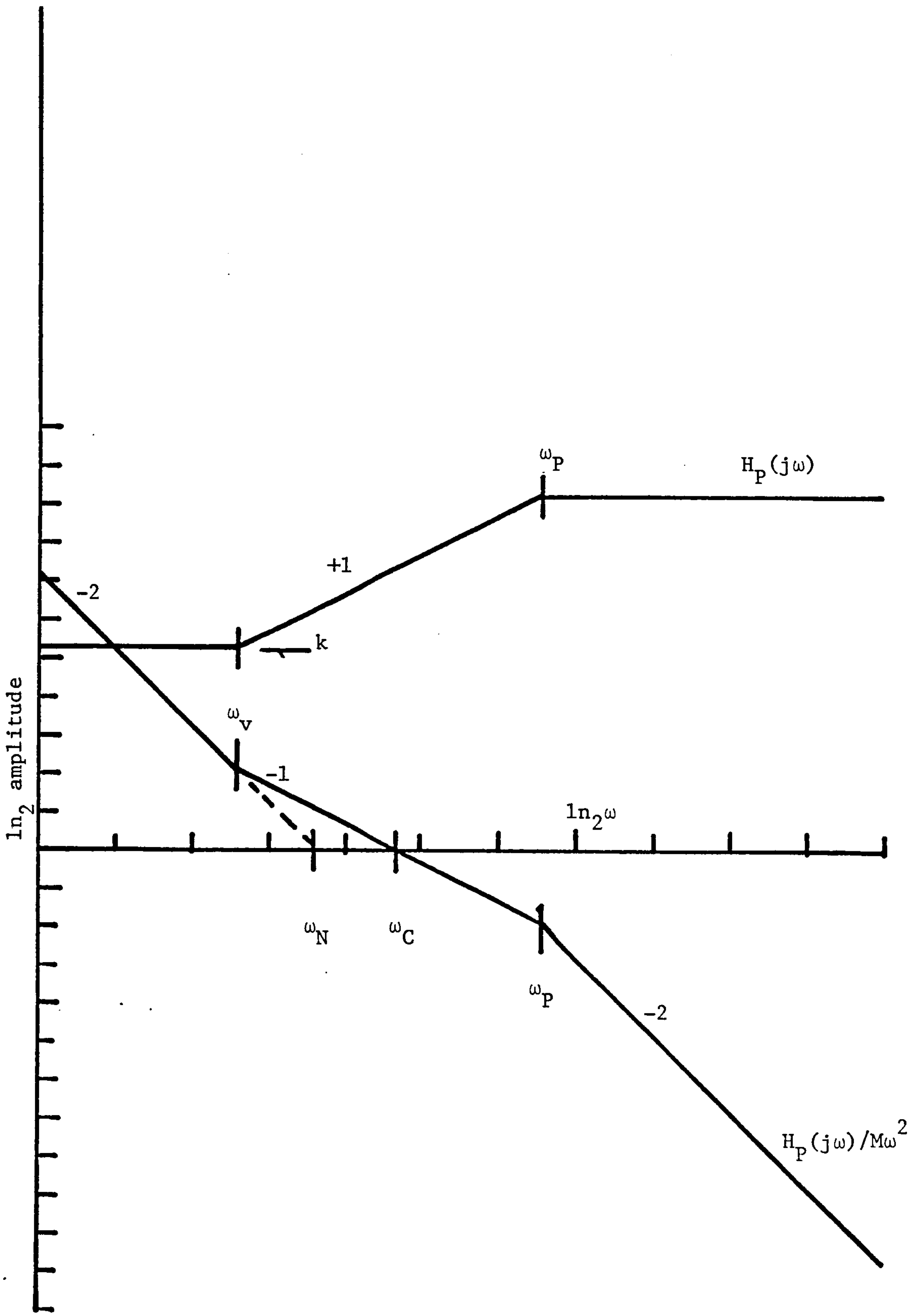
$$\gamma = \omega_P/\omega_V$$

$$= 1/(\tan(45 - \phi_M/2))^2$$

where, from the geometry of the figure 5:

$$\omega_V = \omega_N/\gamma^{1/4}$$

FIGURE 5. BODE PLOTS OF SYNTHETIC SPRING: ANALOG



$$\omega_C = \omega_N \gamma^{1/4}$$

$$\omega_P = \omega_N \gamma^{3/4}$$

Difference Equations for P1-D: From Figure 4, the difference equations must provide the two filters:

$$H_A(s)/K_A K_C = (2\zeta M/K_A K_C)/(1+s/\omega_A)$$

$$= G_A/(s+\omega_A)$$

$$H_P(s)/K_P K_C = (k_s/k_{max})(1+s/\omega_V)/(1+s/\omega_P)$$

$$= (sG_V+G_P/(s+\omega_P)$$

The digital equations for the realizations of these filters are derived using the rectangular rule as follows:

$$x_P(k) = x_P(k) \text{ or } x_L(k)$$

$$u_P(k) = (1-\omega_P T)u_P(k-1) - G_P T x_P(k)$$

$$- G_V\{x_P(k)-x_P(k-1)\}$$

$$u_A(k) = (1-\omega_A T)u_A(k-1) + G_A T x_A(k)$$

$$u(k) = u_P(k) + u_A(k) + x_S(k)$$

It may be noted that the second and third equations could be written as the two equations:

$$u_V(k) = u_V(k-1) - \omega_P T u_P(k-1) - G_P T x_P(k)$$

$$u_P(k) = u_V(k) - G_V x_P(k)$$

where u_V is essentially a state variable. Note that these equations include the input x_S from the signal generator, and the alternative position signal x_L from the LVDT.

Implementation of Program: Appendix A describes a program with two modes of input. They are:

Program P: This program has default parameters, as shown below in parenthesis. New parameters can be entered, and the program can be restarted as Program Q. Values for these parameters are determined as follows:

I_n = Integer value of n used to count cycles.

(= .10X00 = 16)

T = Time interval, musecs.

= 256n

(= .43XF9 = 4096 musecs)

ω_A = Accelerometer break frequency, rads/sec.

$= c/2\zeta M$

(= .48X06 = 36 rads/sec.)

ω_P = Proximeter (or LVDT) break frequency, rads/sec.

$= \gamma^{3/4}\omega_N$

(= .5EX07 = 94.4 rads/sec)

G_A = Accelerometer gain.

$= c/K_A K_C$

(= .66X06 = 50.8)

G_P = Proximeter gain.

$= k_s \omega_P / k_{max}$

$= \gamma^{3/4} M \omega_N{}^3 / K_P K_C$

(= .5EX05 = 23.6)

G_V = Proximeter feedforward gain.

$= G_P / \omega_V$

$= \gamma^{1/4} G_P / \omega_N$

(= .40X03 = 4.0)

The above equations assume that the design damping, c Ns/m, and the required synthetic spring frequency, ω_N rads/sec., are known. Also, n must be chosen so that the program has time to complete a cycle of calculations. As for the default parameters, values for K_A, K_P, and K_C are assumed as discussed earlier, the proof mass M is 0.278 kg., ζ is 1/2, and γ is 16, corresponding to a phase margin ,ϕ_M, of 62 degrees. Default values for n, c, and ω_N are the same as for Program T discussed below.

Program T: In this program, default values are included for the following parameters, and the remainder are calculated from them. They can be entered, and the program can be restarted as Program U:

n = Integer value for n in floating-point format.

(= .40X05 = 16)

c = Design damping, Ns/m.

(= .50X04 = 10 Ns/m)

ω_N = Synthetic spring natural frequency, rads/sec.

= SQRT{k_s/M}, where k_s=design stiffness, N/m.

(= .5EX04 = 11.8 rads/sec, i.e., k_s=38.7 N/m)

Plots of Real Damping and Response Amplitude: Plots of the real damping, Re{H_C}, and the amplitude of the response ratio, Norm{R_C}, are supplied as Figures 6 to 15 for five values of the design damping, c, three values of the design stiffness, k_s, and three values of ζ. Note that the real damping goes negative at low frequencies when $\zeta > 1/2$. Otherwise, the damping is positive over the range of frequencies shown, and is asymptotic to the design damping, c. Although the design stiffness, k_s, was varied over a 16:1 range, it had relatively little effect on the damping curves. Previous investigations, using much lower values for the phase margin, have shown resonance peaks in both curves. Unfortunately, due to the choice of program for the Z80, adequate phase margins could not be used, however, the problem of resonance peaks has been solved since the introduction of floating-point arithmetic.

Timing: The P and T programs described in the Appendix use the #0 and #1 timer interrupt modes available on the 8051 series. The #0

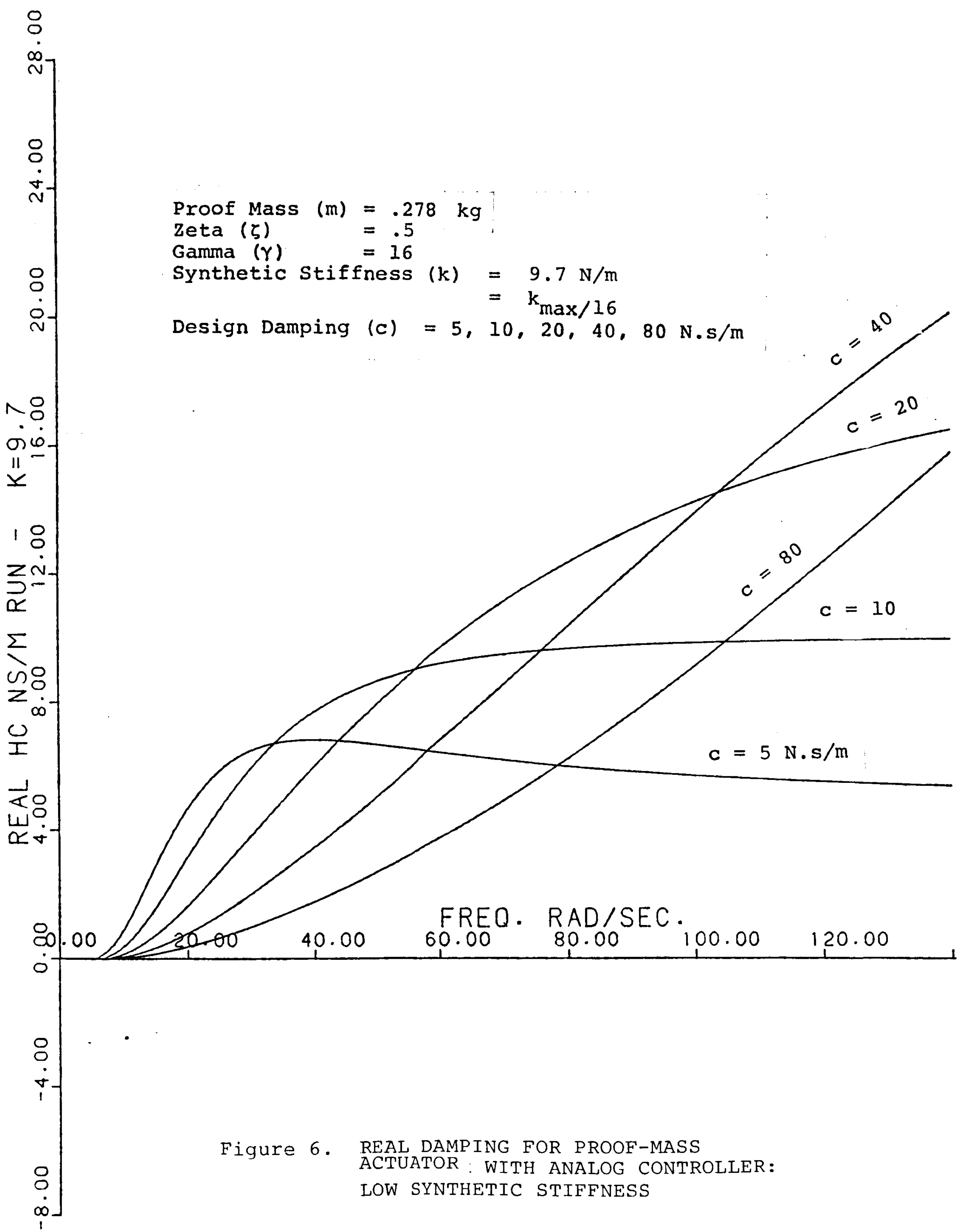


Figure 6. REAL DAMPING FOR PROOF-MASS ACTUATOR WITH ANALOG CONTROLLER: LOW SYNTHETIC STIFFNESS

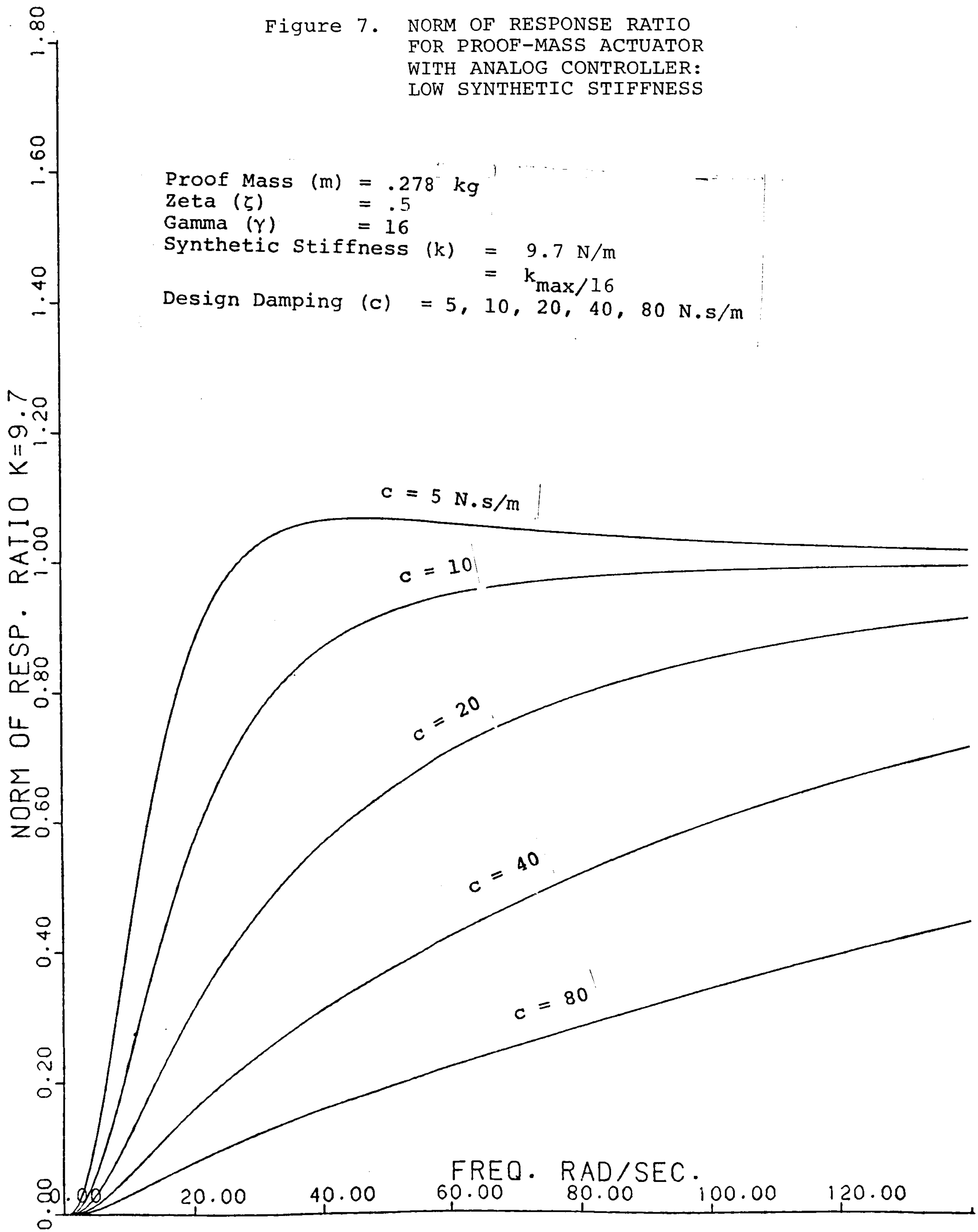


Figure 7. NORM OF RESPONSE RATIO
FOR PROOF-MASS ACTUATOR
WITH ANALOG CONTROLLER:
LOW SYNTHETIC STIFFNESS
Proof Mass (m) = .278 kg
Zeta (ζ) = .5
Gamma (γ) = 16
Synthetic Stiffness (k) = 9.7 N/m
= kmax/16
Design Damping (c) = 5, 10, 20, 40, 80 N.s/m
c = 5 N.s/m
c = 10
c = 20
c = 40
c = 80
NORM OF RESP. RATIO K=9.7
FREQ. RAD/SEC.
0.00
0.20
0.40
0.60
0.80
1.00
1.20
1.40
1.60
1.80
0.00
20.00
40.00
60.00
80.00
100.00
120.00

Figure 8. REAL DAMPING FOR PROOF-MASS ACTUATOR WITH ANALOG CONTROLLER: LOW ACCELEROMETER GAIN

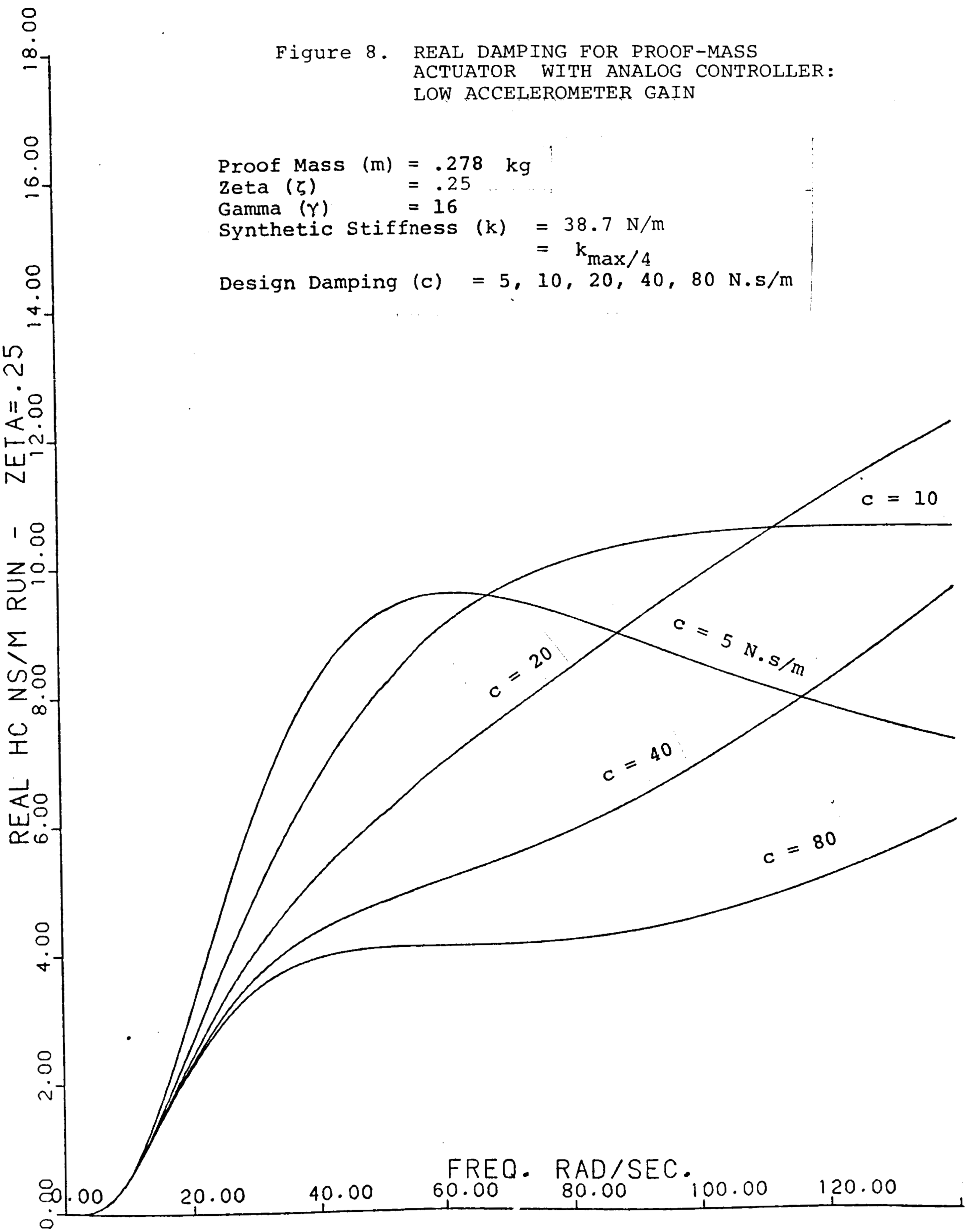


Figure 9. NORM OF RESPONSE RATIO FOR PROOF-MASS ACTUATOR WITH ANALOG CONTROLLER: LOW ACCELEROMETER GAIN

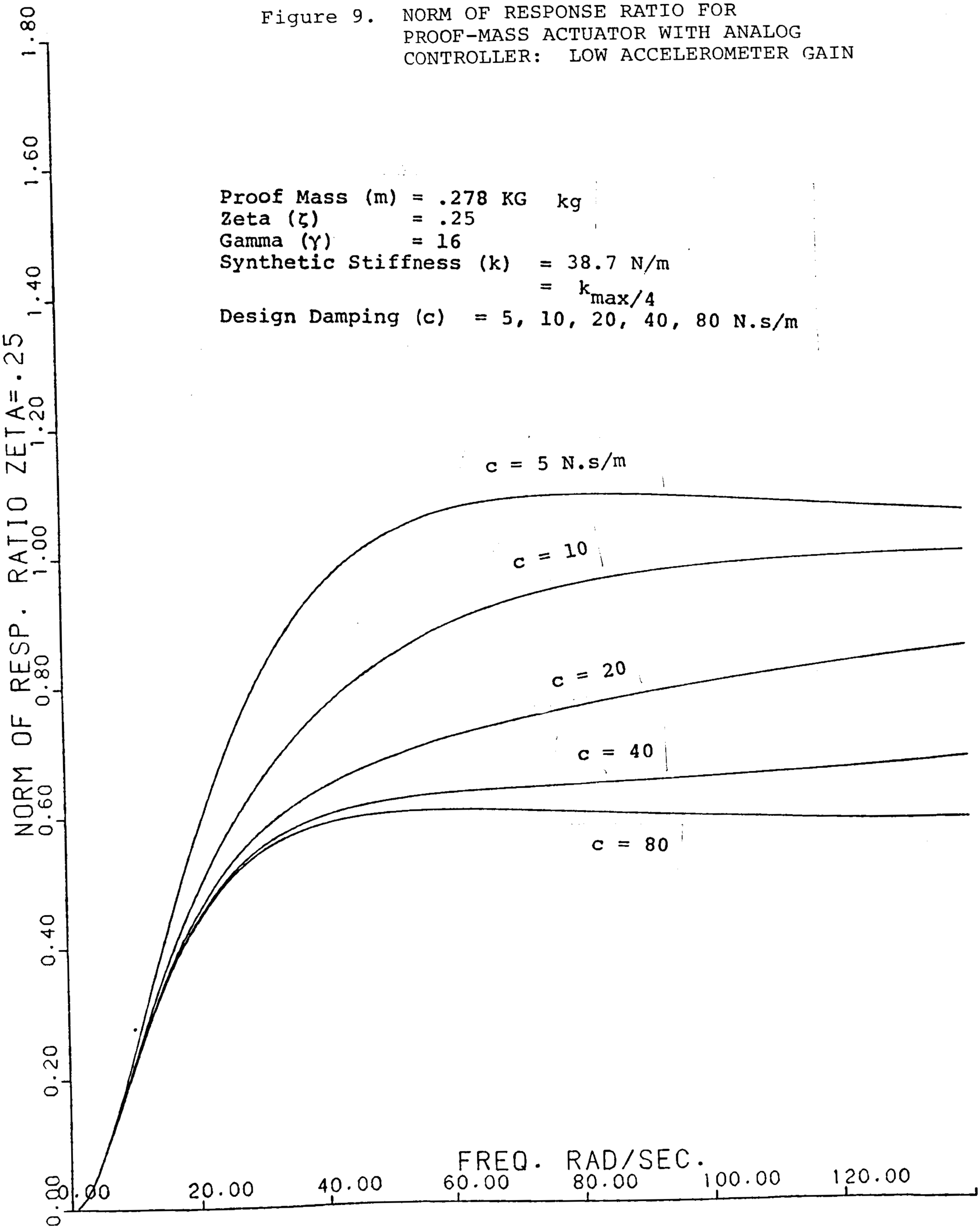
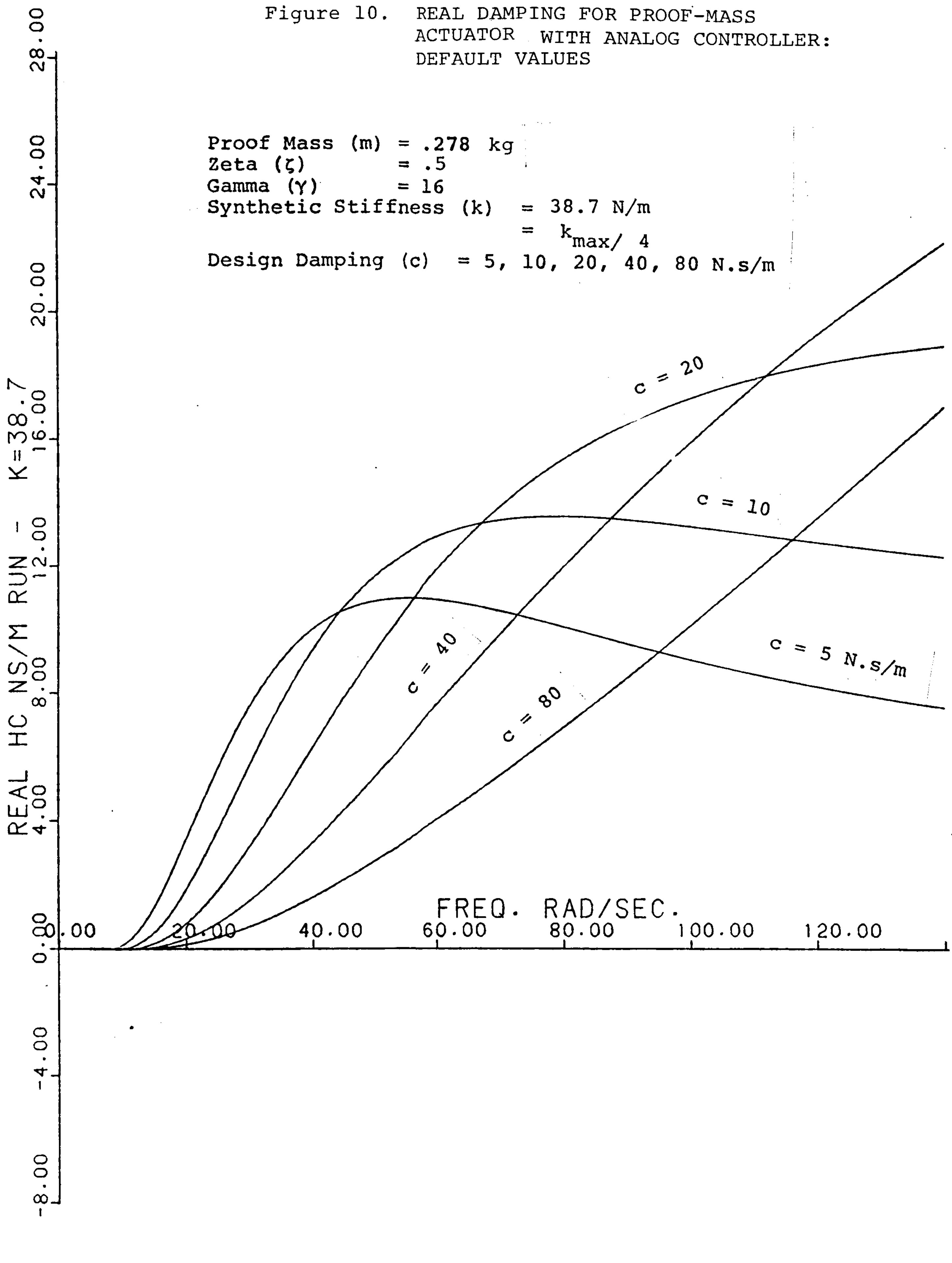


Figure 10. REAL DAMPING FOR PROOF-MASS ACTUATOR WITH ANALOG CONTROLLER: DEFAULT VALUES



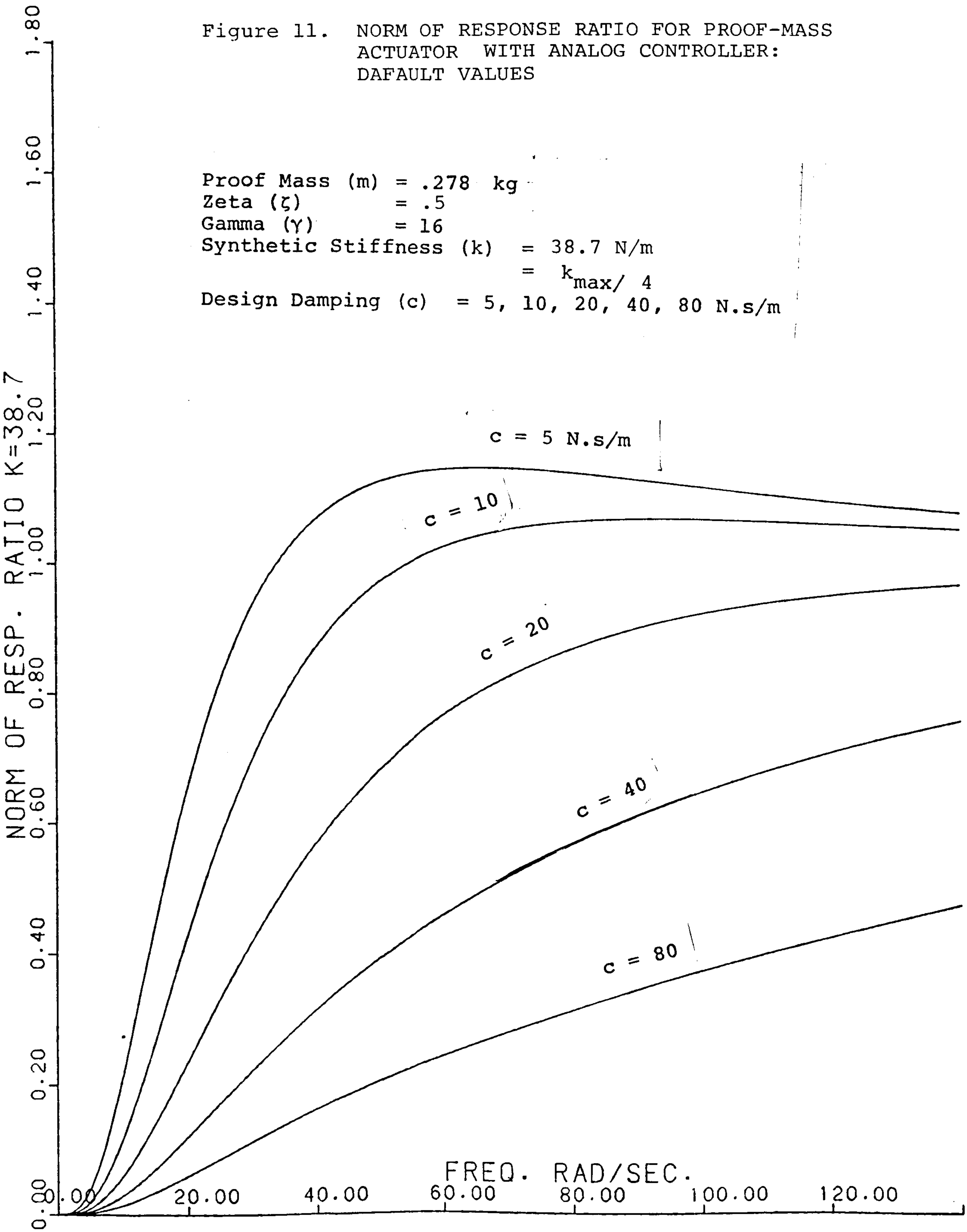


Figure 11. NORM OF RESPONSE RATIO FOR PROOF-MASS ACTUATOR WITH ANALOG CONTROLLER: DAFAULT VALUES

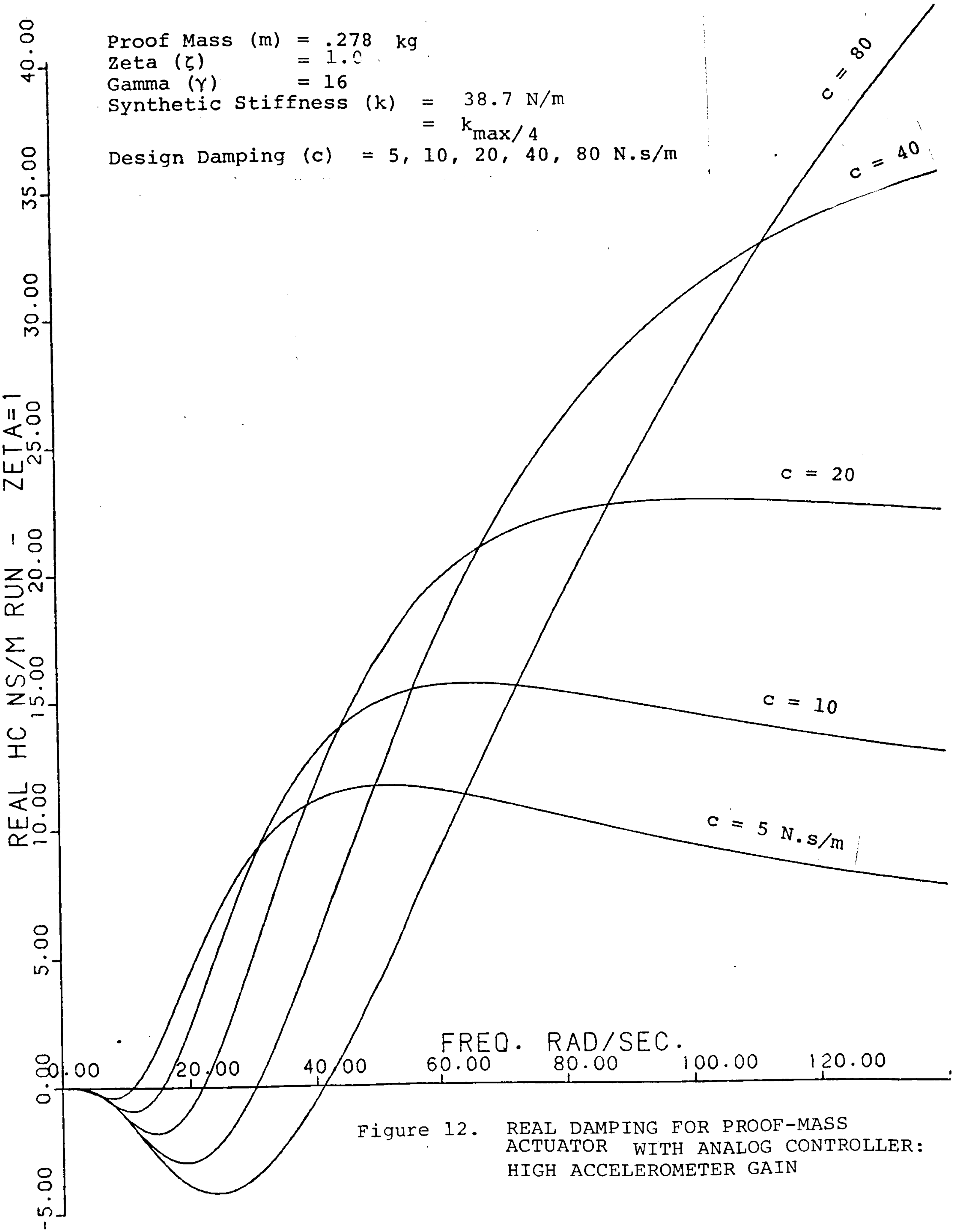
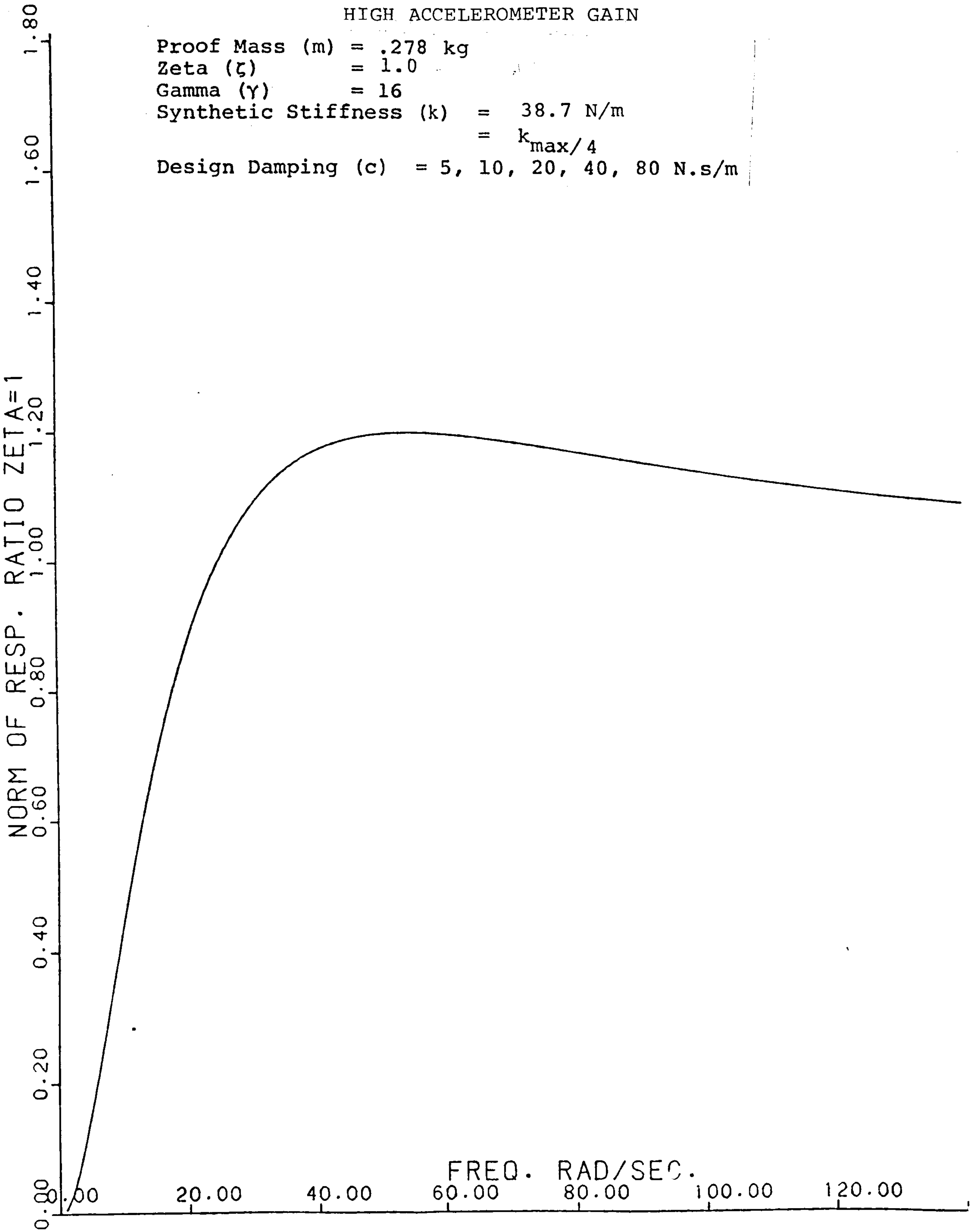


Figure 12. REAL DAMPING FOR PROOF-MASS ACTUATOR WITH ANALOG CONTROLLER: HIGH ACCELEROMETER GAIN

Figure 13. NORM OF RESPONSE RATIO FOR PROOF-MASS ACTUATOR WITH ANALOG CONTROLLER: HIGH ACCELEROMETER GAIN



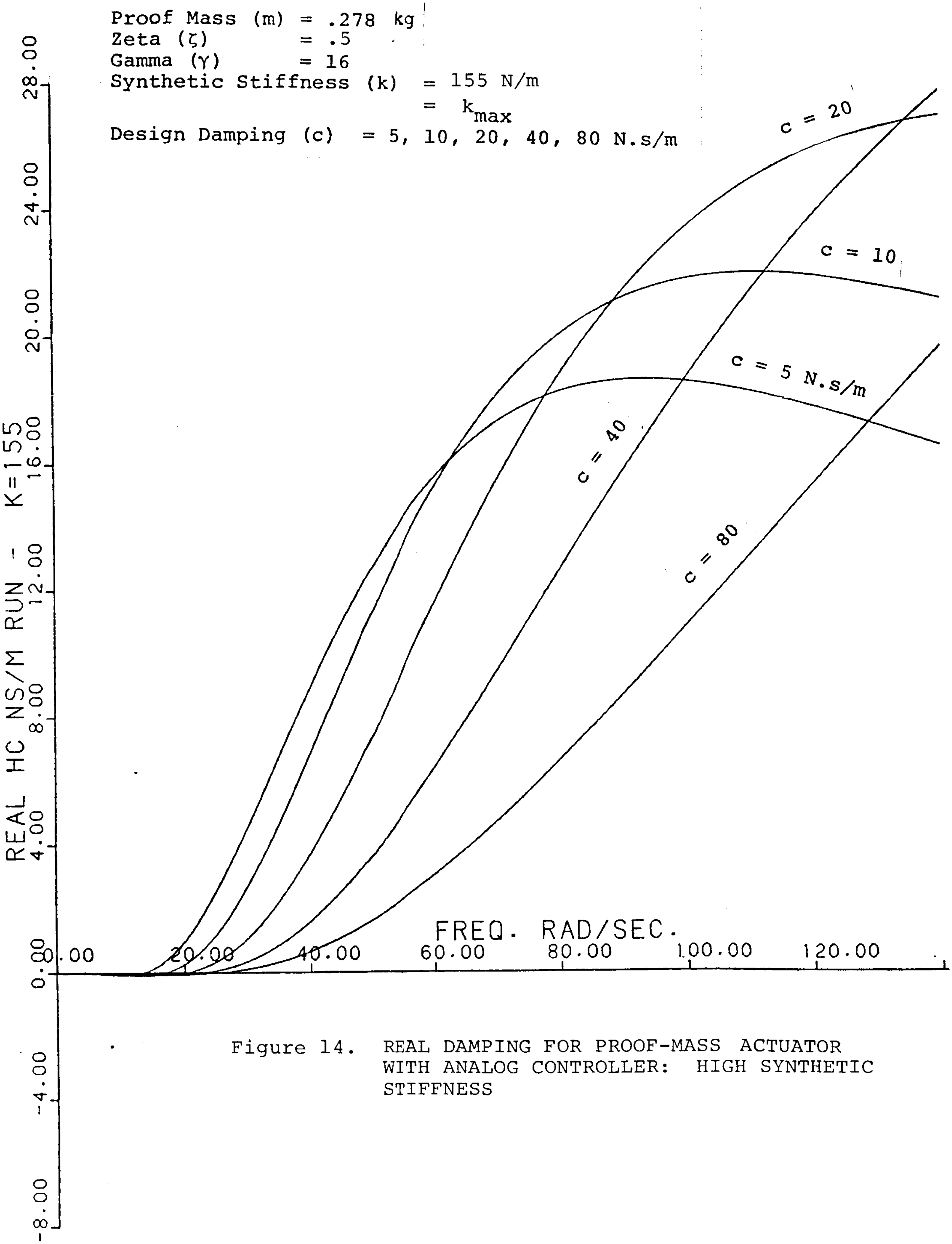
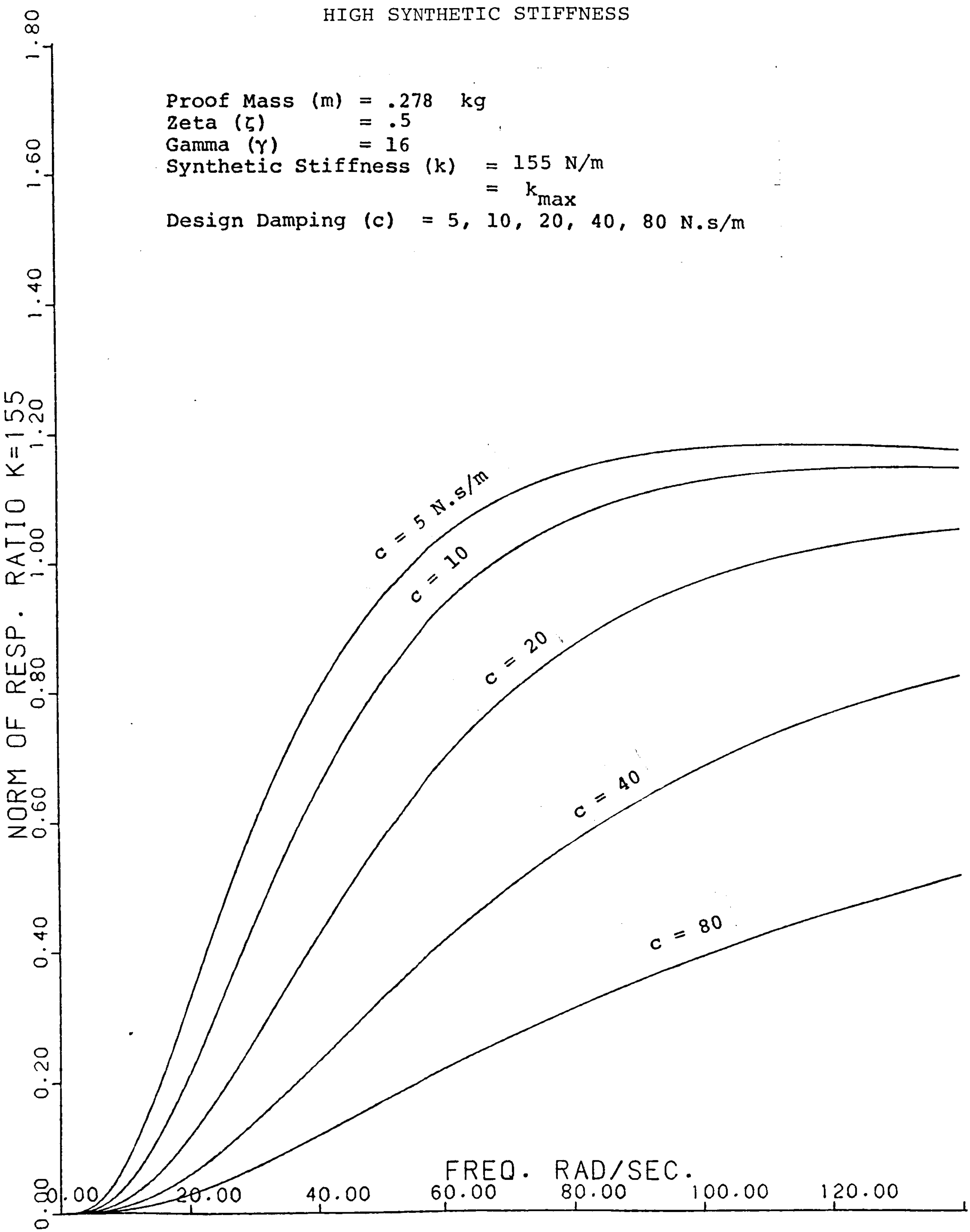


Figure 14. REAL DAMPING FOR PROOF-MASS ACTUATOR WITH ANALOG CONTROLLER: HIGH SYNTHETIC STIFFNESS

Figure 15. NORM OF RESPONSE RATIO FOR PROOF-MASS ACTUATOR WITH ANALOG CONTROLLER: HIGH SYNTHETIC STIFFNESS



interrupt is encountered every 256 musecs, and is used to reset the pulse-width-modulator (PWM), while the #1 interrupt is used to set the pulse width of the PWM. The counter n is used to set the value for T, which is equal to 256n musecs. Typically, n has been 16, resulting in a value for T of 4096 musecs. Lower values, such as 3072 musecs, have been used, but, according to Reference 5, an excessively small value for T can cause problems with round-off noise, even if the digital calculations are completed in time. Both the P and T programs update their output at the end of their cycle, so that there is a full-cycle time-delay of T.

Digital Program by w-Plane Analysis

The control equations described in the preceding paragraphs, and contained in the program described in Appendix A, do not allow for a zero-order-hold, or for the time delay T which is inherent in the method of calculation. Typically, an analog plant driven by a digital filter can be represented by Figure 16a, if there is no time delay , and by Figure 16b if there is a time delay, following methods described in the literature, such as for example, in References 5 or 6. The zero-order hold has a z-transform equal to (z-1)/z, thus the open loop transfer functions are:

for no delay:

$$U(z)/X(z) = H(z)((z-1)/z)Z\{G(s)/s\}$$

and for a delay of T:

$$U(z)/X(z) = H(z)((z-1)/z^2)Z\{G(s)/s\}$$

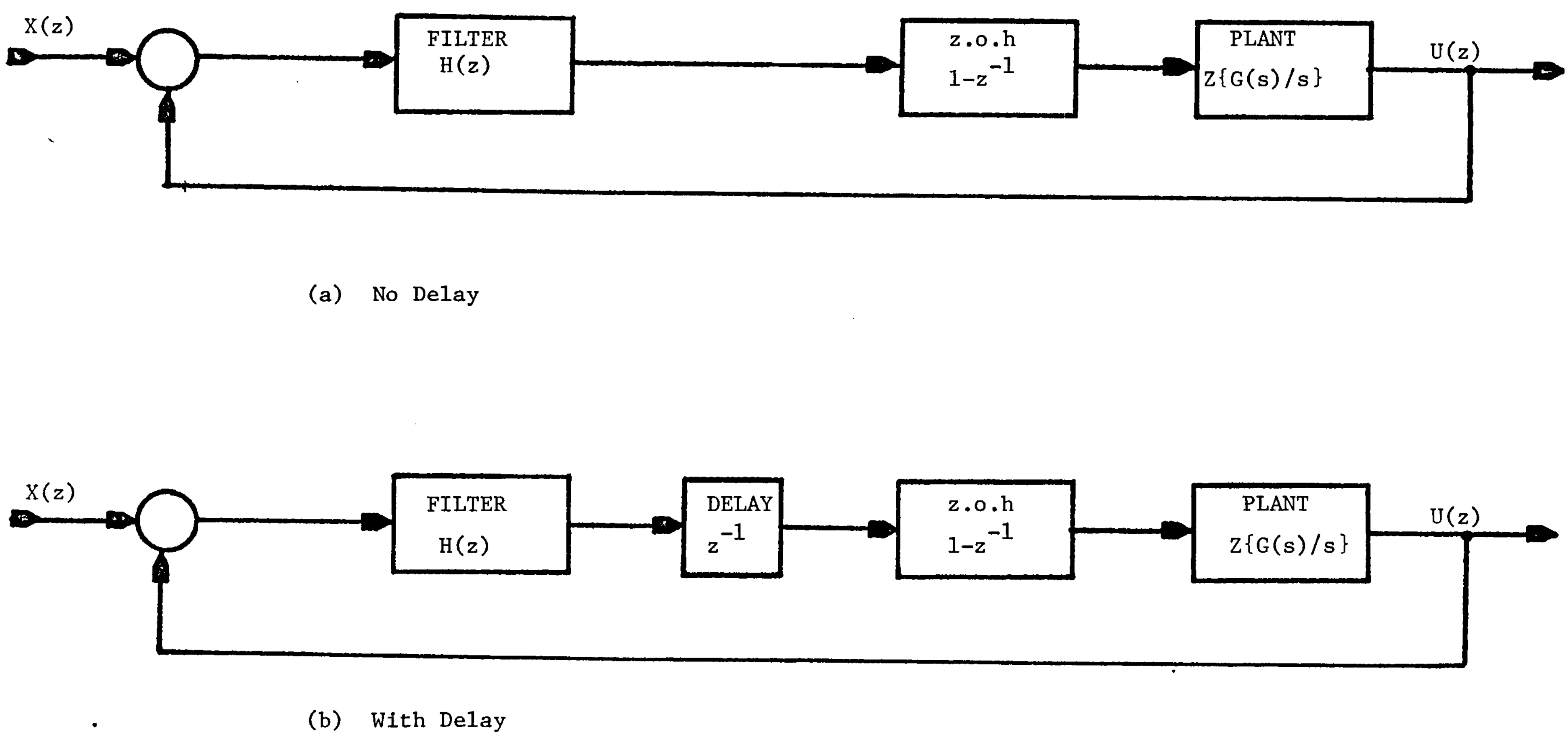


FIGURE 16. DIGITAL FILTER WITH ANALOG PLANT: EFFECT OF DELAY

where Z represents the z-transform equivalent of a Laplace transform. A block diagram of the damper, corresponding to Figure 4, but incorporating the concepts shown in Figure 16, is shown in Figure 17. The input is represented as an acceleration a_F, so that the z-transform derived below represents $F(z)/A_F(z)$, which can be readily converted to the form H_C. First, two equations are derived from Figure 16:

$$F(z) = \{H_A(z)A_F(z) - H_P(z)X_D(z)\}/z$$

$$X_D(z) = \{F(z)/M - A_F(z)\}((z-1)/z)Z\{1/s^3\}$$

so that the overall transfer-function can be written as:

$$F(z)/A_F(z) = \{D_A(z) + MD_P(z)\}/\{1 + D_P(z)\}$$

Note that:

$$Z\{1/s^3\} = T^2 z(z+1)/2(z-1)^3$$

then:

$$D_A(z) = H_A(z)/z$$

$$D_P(z) = H_P(z)((z-1)/z^2)Z\{1/Ms^3\}$$

$$= H_P(z)(T^2/2M)(z+1)/z(z-1)^2$$

The w-transform maps the z-plane into a space which more nearly resembles the s-plane. In fact, as s moves along the imaginary axis from zero to the Nyquist frequency, as represented by $s=j\omega$, w moves along the real axis from zero to infinity, as represented by $w=j\nu$. The substitution for z is:

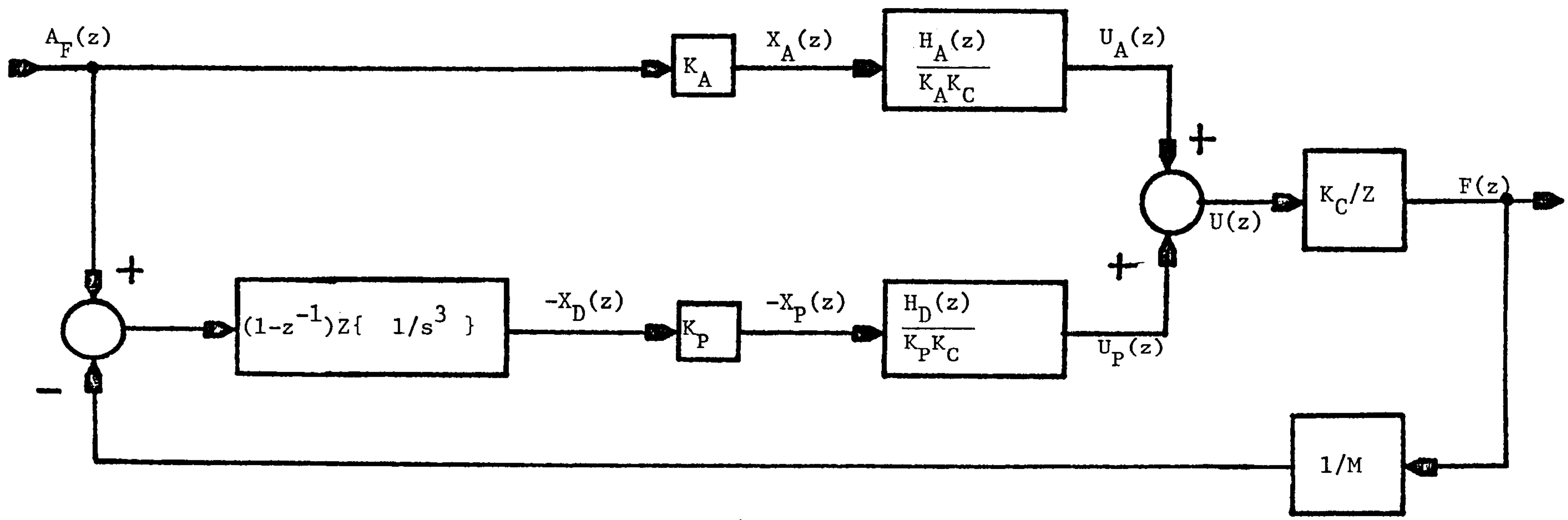


FIGURE 17. PROOF-MASS ACTUATOR CONTROLS: DIGITAL

$$z = \{1 + wT/2\}/\{1 - wT/2\}$$

while ν is given by:

$$\nu = (2/T)\tan\{\omega T/2\}$$

the inverse being given by:

$$\omega = (2/T)\arctan\{\nu T/2\}$$

so that the D(w) transfer functions become:

$$D_A(w) = H_A(w)\{1 - wT/2\}/\{1 + wT/2\}$$

$$D_P(w) = H_P(w)\{1 - wT/2\}^2/Mw^2\{1 + wT/2\}$$

Design in the w-Plane: The rules for designing in the w-plane are almost identical to those for designing in the s-plane. The familiar Bode plots can be made, the only difficulty being that the w-transfer functions are often not of minimum phase form. This means that the phase cannot be inferred from the Bode plot alone, but this is not a problem of any significance. The Bode plots for $D_A(w)$ and $D_P(w)$ are shown in Figures 18 and 19. In the following discussion, T=4096 musecs, so that the Nyquist frequency is $\pi/T=$ 767 rads/sec in the s plane. Proceeding with the design in the w-plane, using almost identical methods to those used in the s-plane, but taking $\zeta=1/2$, we find that:

$$H_A(w) = M/\{1+w/\nu_A\}$$

$$H_P(w) = k_s\{1+w/\nu_V\}/\{1+w/\nu_P\}$$

Using the previous default values of n (=16), c (=10 Ns/m),

FIGURE 18. BODE PLOT OF SYNTHETIC DAMPER: DIGITAL

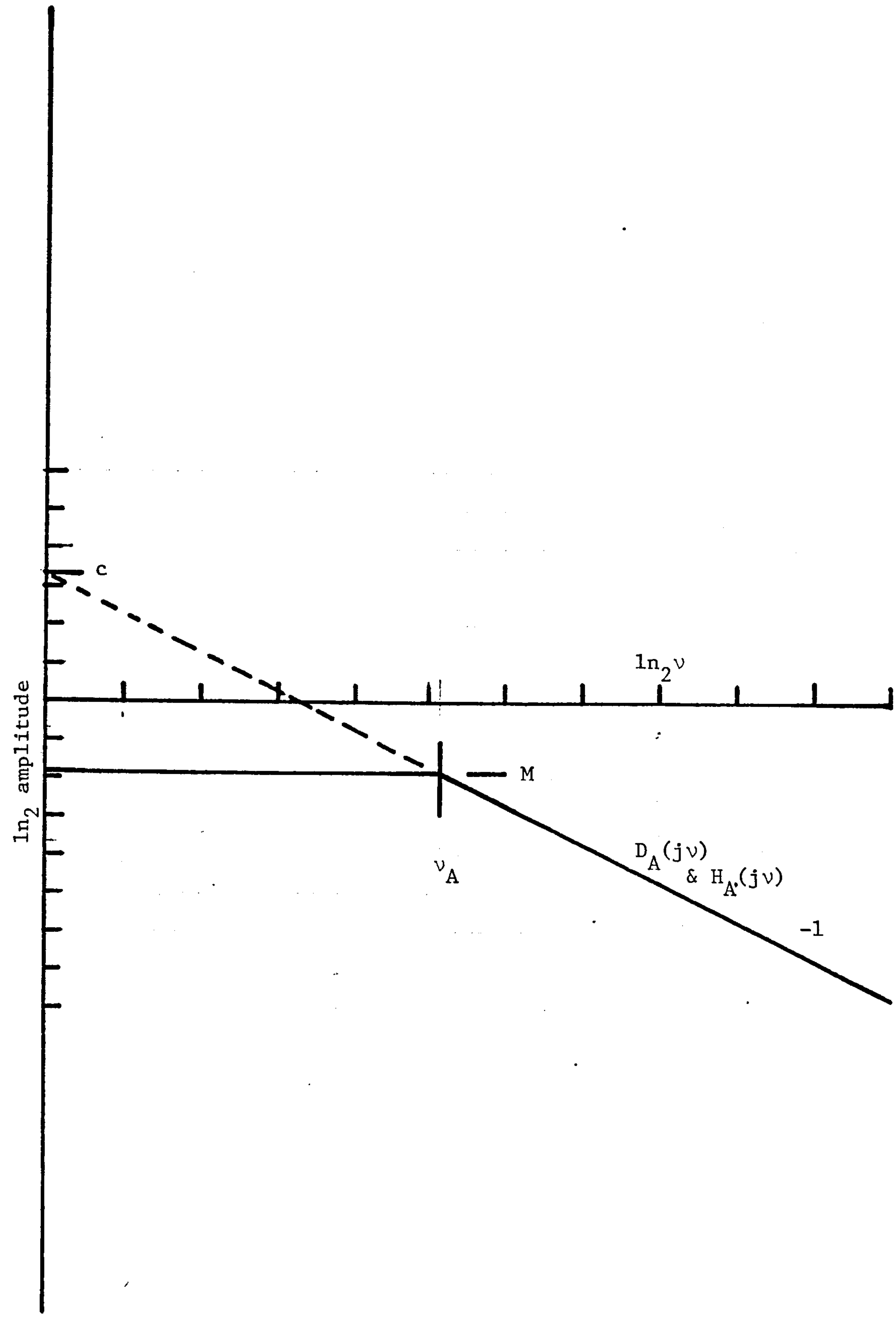
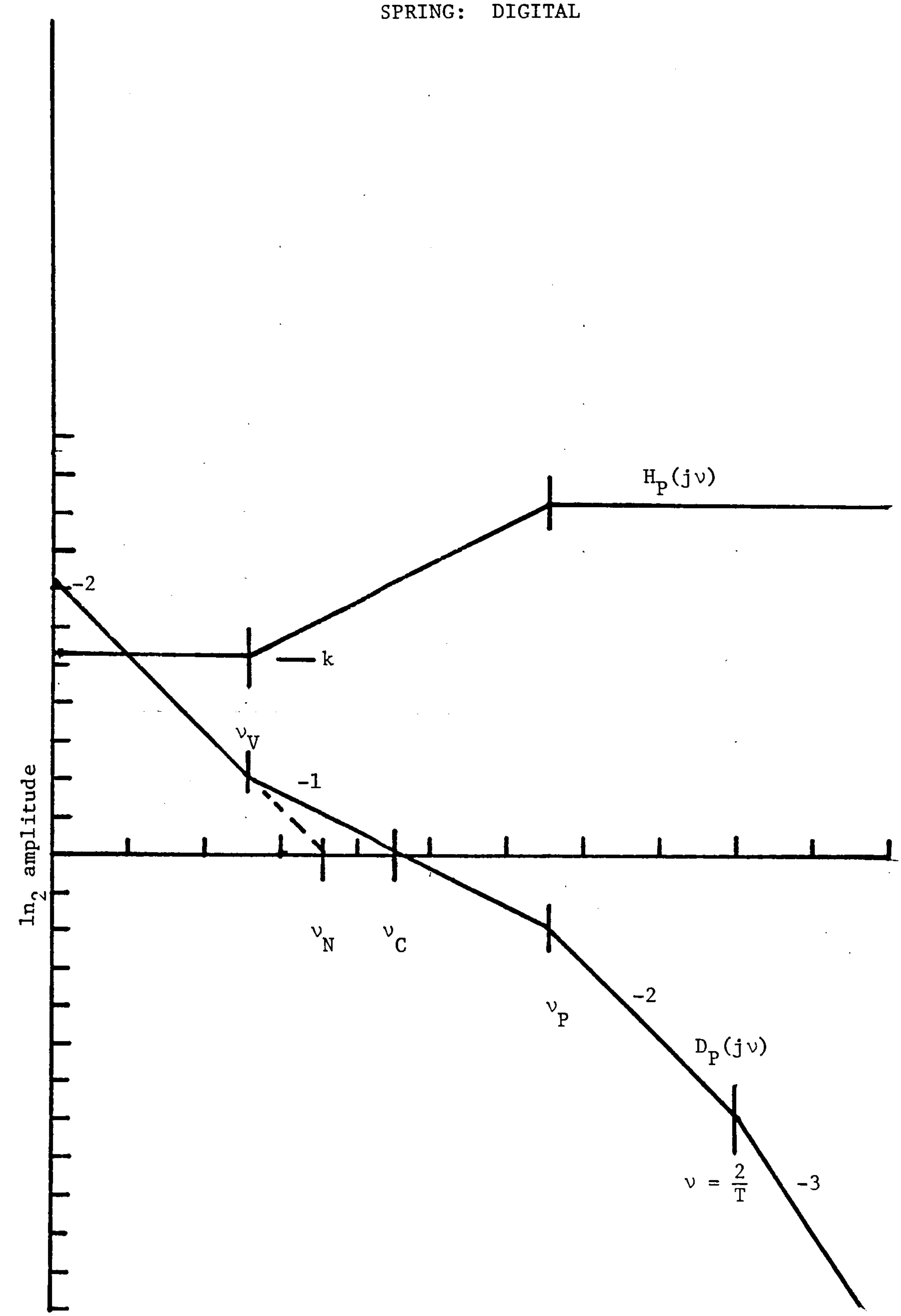


FIGURE 19. BODE PLOTS OF SYNTHETIC SPRING: DIGITAL



and k_s (=38.7 N/m), and taking γ=16, we find values for ν_A, ν_P, and ν_V which are numerically equal to the corresponding ω values found for the analog design case. Thus the actual ω values have decreased according to the transformation law given above.

The apparent -1 break at ν=2/T=488.3 rads/sec in the Bode plot of $D_P(w)$ is misleading, because it is a multiple phase break and introduces additional phase lags of 90 degrees in the case of D_A and 135 degrees in the case of D_P. For D_A to have the high frequency performance characteristic of a damper, it should lag 90 degrees. However, calculations show that the lag is 180 degrees, so that real damping is zero, at ν= 525 rads/sec, corresponding to a true frequency of 401 rads/sec. Again, although the value for the gain at the design crossing frequency ν_C (= 23.6 rads/sec.) of the open-loop transfer function D_P is calculated to be 0.9996, the phase margin is found to be 8.4 degrees less than the design value of 62 degrees, because of the triple phase break at T/2

<u>Derivation of the Difference Equations</u>: To transform back to the z-plane, we apply the transformation:

$$w = (2/T)(z-1)/(z+1)$$

From Figure 16, the difference equations must provide two filters which, on transformation to the z-plane, become:

$$H_A(z)/K_AK_C = G^*{}_A(z+1)/(z-k_A)$$

$$H_P(z)/K_PK_C = G^*{}_P(z-k_V)/(z-k_P)$$

Values for the new terms are as follows, with numerical default values in parenthesis:

$$G^{*}_{A} = (cT/2K_AK_C)/(1+\nu_AT/2)$$

(= 0.194/2)

$$k_A = (1-\nu_AT/2)/(1+\nu_AT/2)$$

(= 0.863)

$$G^{*}_{P} = (k\nu_P/k_{max}\nu_V)(1+\nu_VT/2)/(1+\nu_PT/2)$$

(= 3.39)

$$k_V = (1-\nu_VT/2)/(1+\nu_VT/2)$$

(= 0.976)

$$k_P = (1-\nu_PT/2)/(1+\nu_PT/2)$$

(= 0.676)

The difference equations derived from the above are:

$$u_P(k) = k_Pu_P(k-1) - G^{*}_{P}(1-k_V)\{x_P(k)+x_P(k-1)\}/2$$
$$- G^{*}_{P}(1+k_V)\{x_P(k)-x_P(k-1)\}/2$$

$$u_A(k) = k_Au_A(k-1) + G^{*}_{A}\{x_A(k)+x_A(k-1)\}$$

These can be compared with the equations obtained by the rectangular rule from the analog design;

$$u_P(k) = (1-\omega_PT)u_P(k-1) - G_PTx_P(k)$$
$$-G_V\{x_P(k)-x_P(k-1)\}$$

$$u_A(k) = (1-\omega_A T)u_A(k-1) + G_A T x_A(k)$$

It will be noted that the w-plane design method directly implies use of the trapezoidal rule. It is easier to compare the two approaches if the default values are substituted for the coefficients. For the w-plane design, we get:

$$u_P(k) = 0.676u_P(k-1) - 0.0814\{x_P(k)+x_P(k-1)\}/2$$

$$- 3.35\{x_P(k)-x_P(k-1)\}$$

$$u_A(k) = 0.863u_A(k-1) +0.194\{x_A(k)+x_A(k-1)\}/2$$

which can be compared with the results of the rectangular rule design:

$$u_P(k) = 0.613u_P(k-1) - 0.0967x_P(k) - 4.0\{x_P(k)-x_P(k-1)\}$$

$$u_A(k) = 0.853u_A(k-1) + 0.208x_A(k)$$

The worst difference between the coefficients used in the two sets of equations is about 20 percent, so that, evidently, there is no serious loss of performance with the rectangular rule. The difference equations for the w-plane design can be put into more useable form, and the damping can be calculated readily from its w-transform. However, we shall look into another point first.

System with Minimum Delay

Numerical Accuracy: Consider first, that only the accelerometer circuit is active. Then a +1 input, representing 9.8 m/s^2 if the channel is calibrated, should result in a force on the proof-mass of Mg = 2.72 N, or an output from the computer of $Mg/K_C = 1.42$, which is out of the range of the system. As a check on numerical accuracy, let $u_A(k-1)$ equal 1.42, and let $x_A(k)$ equal 1.0, then:

$$\begin{aligned} u_A(k) &= (0.863)(1.42) + (0.194) \\ &= 1.42 \\ &= u_A(k-1) \end{aligned}$$

However, the output is quantized to only 256 values, so that the maximum input of +1 is equivalent to (0.194)(256) = 49 quantized values. In other words, there are only 49 possible values for the coil force in the static case, and one third of them are out of range. Looking at the synthetic spring from the same approach, an input of -1, representing the proof mass against the structure, should result in an output of 0.25, representing one quarter of k_{max}. Taking $u_V(k-1)$ equal to 0.25, and $x_P(k)$ equal to -1, we get:

$$\begin{aligned} u_P(k) &= (0.676)(0.25) - (.0814)(-1) \\ &= 0.25 \\ &= u_P(k-1) \end{aligned}$$

In this case, however, the input is equivalent to (0.0814)(256) = 20 values, so that the restoring force is limited to 20 quantized values. It is somewhat surprising that the synthetic spring appears to be smooth to the touch, however, it might prove impossible to obtain a very small spring value, equal to a few percent of k_{max}. This quantization effect would be reduced if T were increased, but the phase margin might also be reduced at the same time.

Minimum Delay: Figure 20 shows an analog plant driven by a digital filter in which the time delay is kept to a minimum by timing the output to occur immediately after the calculations are completed. The basic period T_0 is assumed to be 256 musecs, but calculations are repeated every T ($=nT_0$) musecs, while output occurs at mT_0, with m<n. We now have the open-loop transfer function:

$$U(z^n)/X(z^n) = H(z^n)((z^n-1)/z^{n+m})Z_n\{G(s)/s\}$$

The delay and zero-order hold blocks of Figure 17 can be modified accordingly, so that the overall transfer function becomes:

$$F(z^n)/A_F(z^n) = \{D_A(z^n)+MD_P(z^n)\}/\{1+D_P(z^n)\}$$

where:

$$D_A(z^n) = H_A/z^m$$

$$D_P(z^n) = H_P(z^n)((z^n-1)/z^{n+m})Z_n\{1/Ms^3\}$$

$$= H_P(z^n)(T^2/2M)(z^n+1)/z^m(z^n-1)^2$$

The w-transform is now:

$$z^n = (1+wT/2)/(1-wT/2)$$

with its inverse:

$$w = (2/T)(z^n-1)/(z^n+1)$$

also:

$$\omega = (2/T)\arctan(\nu T/2)$$

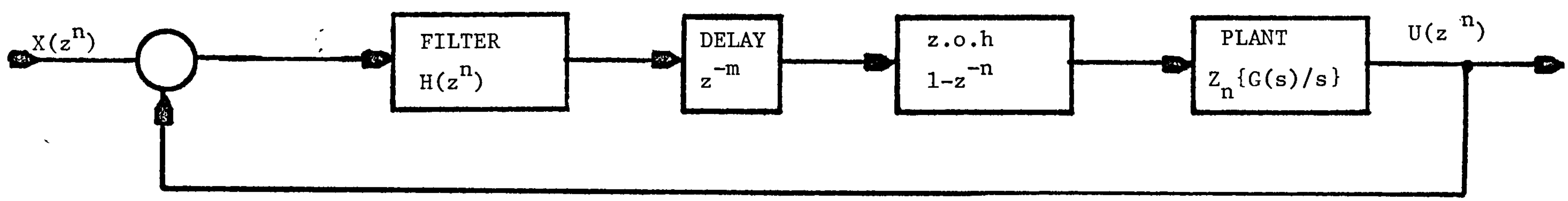


FIGURE 20. DIGITAL FILTER WITH ANALOG PLANT: IMMEDIATE OUTPUT

and:

$$\nu = (2/T)\tan(\omega T/2)$$

The D(w) transfer functions now become:

$$D_A(w) = H_A(w)(\{1-wT/2\}/\{1+wT/2\})^{m/n}$$

$$D_P(w) = H_P(w)\{1-wT/2\}^{(1+m/n)}/Mw^2\{1+wT/2\}^{m/n}$$

Thus, apart from a change in output timing, and possible redesign for improved phase margin, the difference equations are essentially unchanged when the output is speeded up. However, there should be an improvement in the real damping as m/n is decreased, which would partially offset the effect of increasing n to obtain longer cycle times.

<u>Difference Equations for Minimum Delay Case</u>: Assuming that the $H_A(w)$ and $H_P(w)$ filters are essentially the same as before, we find that on applying the inverse w-transform we have $H_A(z^n)$ and $H_P(z^n)$. However, in obtaining the difference equations from these, we obtain expressions for $u_V(nkT_0)=u_V(kT)$, etc., so that the final equations are the same as before. The form in which the equations were left is not the most convenient, but note that the first order transfer function:

$$u(z)/x(z) = a_0(1+z^{-1}a_1)/(1+z^{-1}b_1)$$

can either be written as:

$$u(k) = -b_1u(k-1) + a_0x(k) + a_0a_1x(k-1)$$

or as the pair of equations:

$$u_1(k) = -b_1u_1(k-1) - (a_0/b_1)x(k)$$

$$u(k) = (a_1-b_1)u_1(k) + (a_0a_1/b_1)x(k)$$

where the additional variable, u_1 is essentially a state variable. Using this representation, the complete set of equations can be written as:

$$x_P(k) = x_P(k) \text{ or } x_L(k)$$

$$u_V(k) = k_Pu_V(k-1) - (G^*{}_P/k_P)x_P(k)$$

$$u_P(k) = (k_P-k_V)u_V(k) - (G^*{}_Pk_V/k_P)x_P(k)$$

$$u_B(k) = k_Au_B(k-1) + (G^*{}_A/k_A)x_A(k)$$

$$u_A(k) = (1+k_A)u_B(k) - (G^*{}_A/k_A)x_A(k)$$

$$u(k) = u_P(k) + u_A(k) + x_S(k)$$

where u_V, u_B are the corresponding state variables.

<u>Plots of Real Damping</u>: The real damping can be calculated as:

$$H_c = \text{REAL}\{j\omega F(z)/A_F(z)\}$$

This is shown in Figures 21 to 24 for four cases each. One represents the analog approximation obtained by taking T=0 and is identical to the results shown in Figure 9 for the same parameters, while the remaining three cases are for T=4096, 8192, and 16,384 microseconds. The other parameters which are varied are the output time delay, which is 0 and 4096 microseconds, (m=0,16), and the design damping, which is 10 and 80 Ns/m. The cases where T and the time delay are both 4096 microseconds

FIGURE 21

Real Damping for Proof-Mass Actuator with Digital Controller

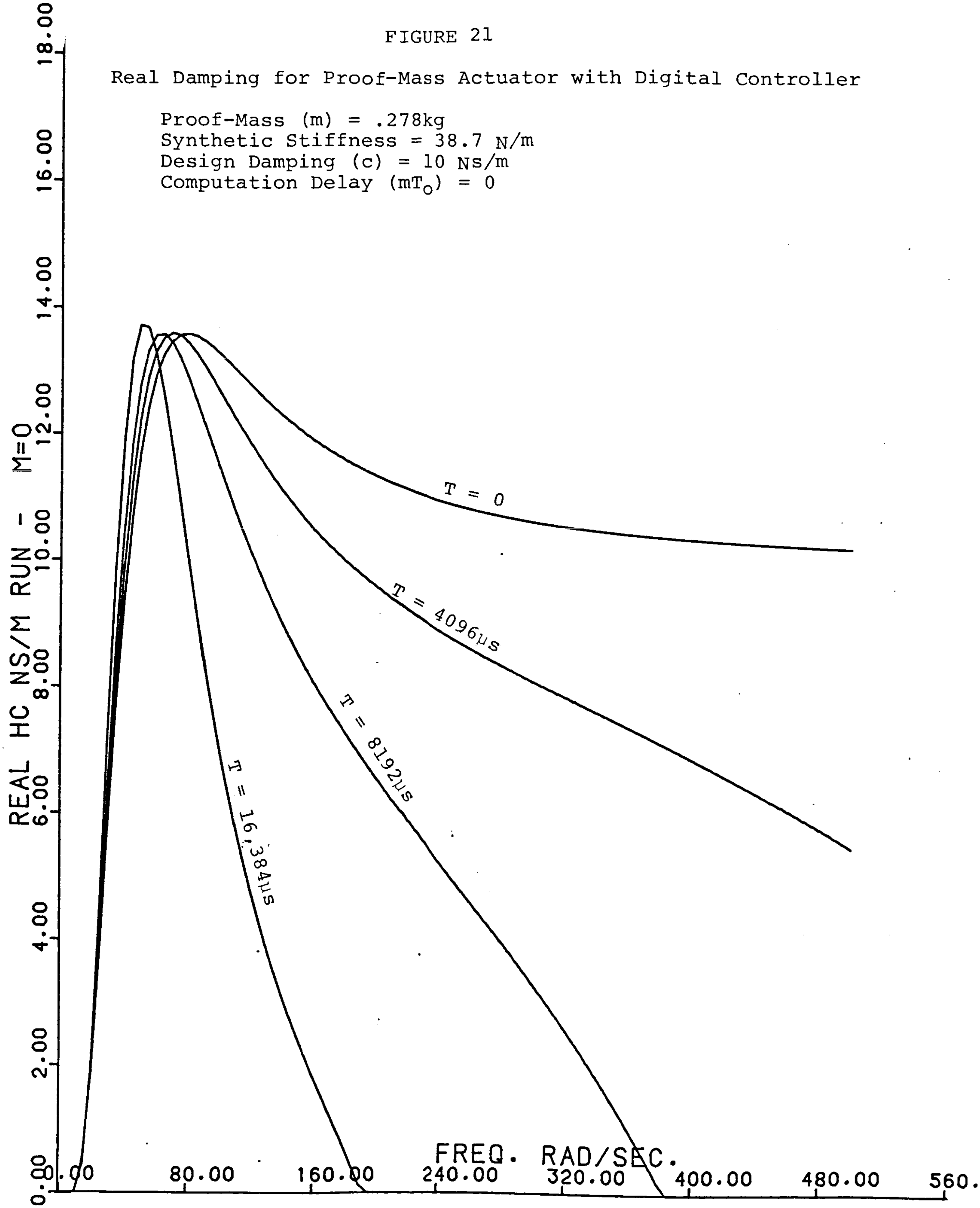


FIGURE 22

Real Damping for Proof-Mass Actuator with Digital Controller

Proof Mass (m) = .278kg
Synthetic Stiffness = 38.7 N/m
Design Damping (c) = 10 Ns/m
Computation Delay (mT_o) = 4096μs

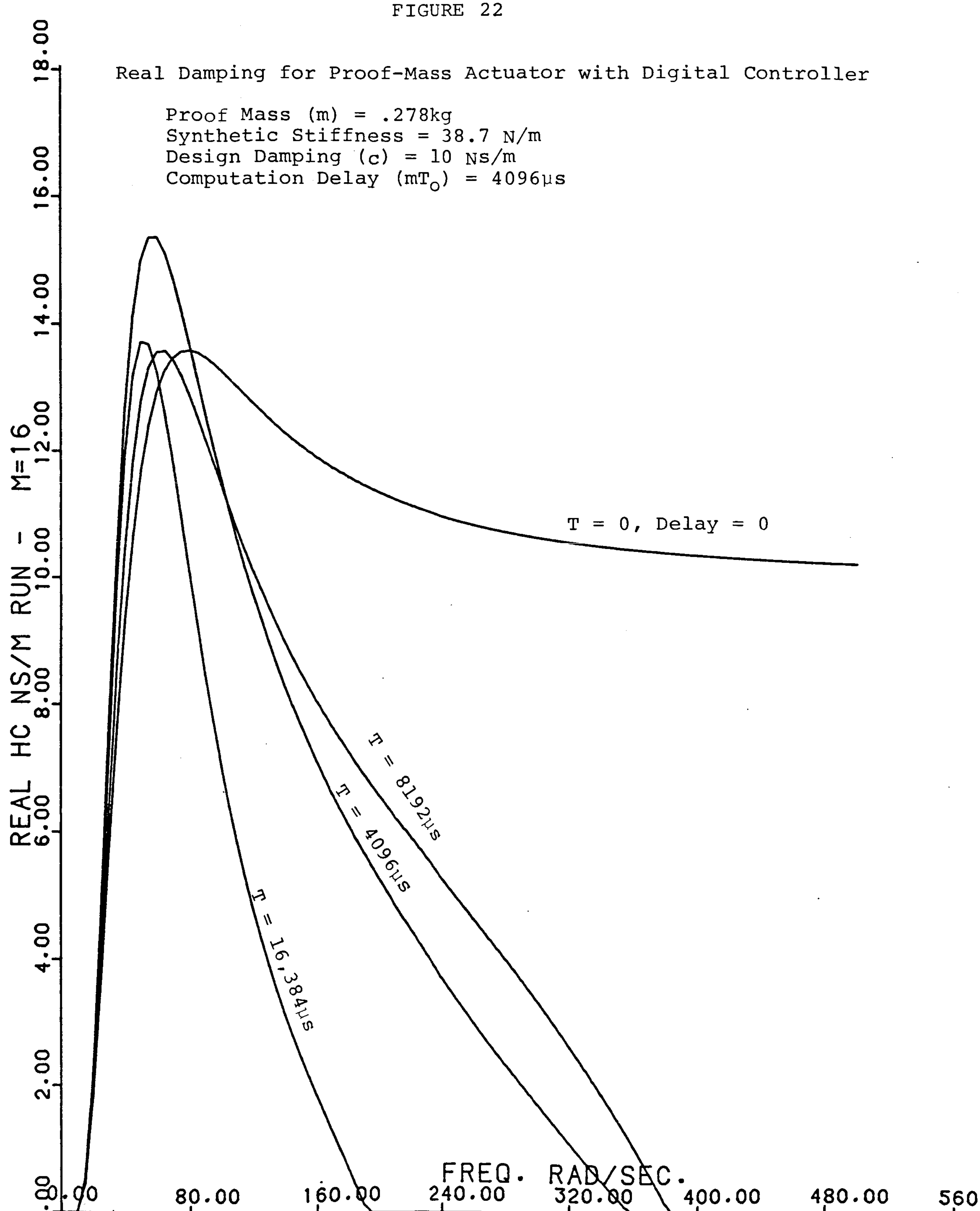
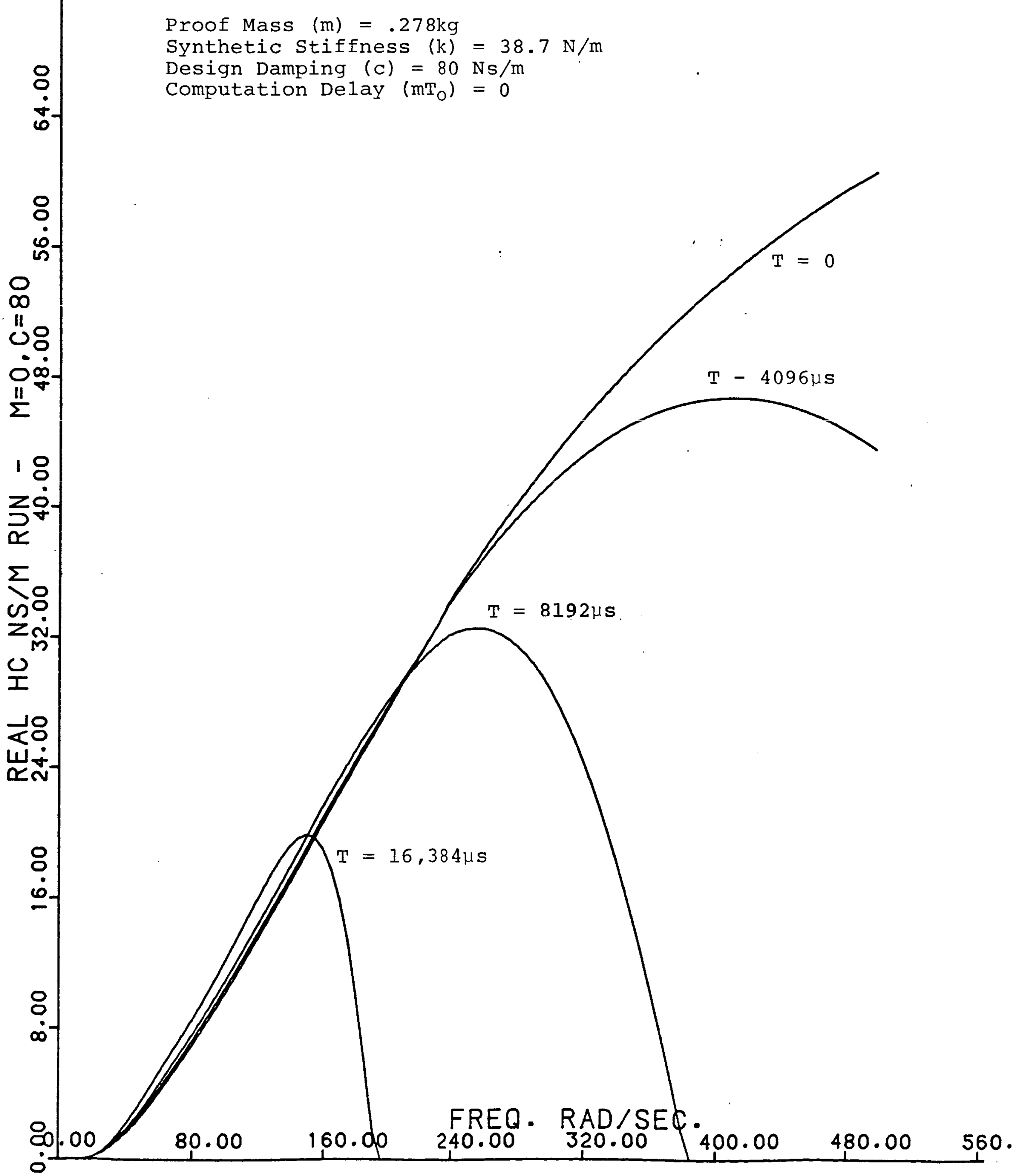


FIGURE 23

Real Damping for Proof-Mass Actuator with Digital Controller



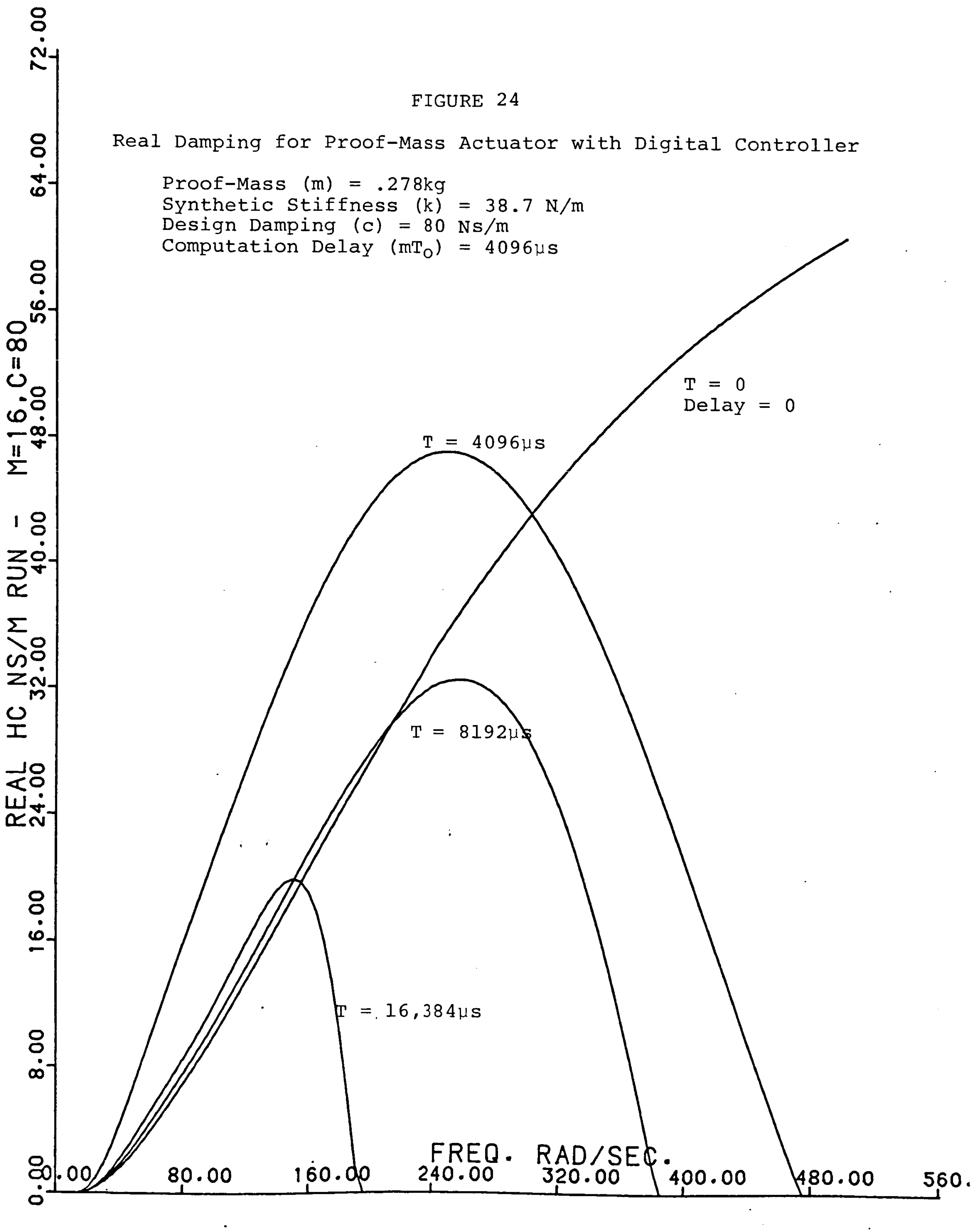


FIGURE 24
Real Damping for Proof-Mass Actuator with Digital Controller
Proof-Mass (m) = .278kg
Synthetic Stiffness (k) = 38.7 N/m
Design Damping (c) = 80 Ns/m
Computation Delay (mT0) = 4096μs
T = 0
Delay = 0
T = 4096μs
T = 8192μs
T = 16,384μs
REAL HC NS/M RUN - M=16,C=80
FREQ. RAD/SEC.
0.00
8.00
16.00
24.00
32.00
40.00
48.00
56.00
64.00
72.00
0.00
80.00
160.00
240.00
320.00
400.00
480.00
560.

corresponds to the P- and T-programs listed in Appendix A.

As might be expected, better agreement with the analog approximation is shown when the time delay is 0. Otherwise, agreement is best when T is a minimum. However, at low frequencies, the higher values for T show increased damping, presumably because of greater phase lags. It must be emphasized that two of the timing cases, where the time delay is zero or equal to T, have accurate solutions. The remaining cases introduce additional approximations of uncertain validity.

SUMMARY

Controller Design: The third in a series of controllers for the UVA Proof-Mass Actuator has been designed, built in prototype form, and demonstrated. The present design uses an INTEL 8031 microcontroller mounted in an SDK-51 System Design Kit. Previously, an analog controller had been breadboarded, and a Z80 controller had been developed as a slave to a TRS80 computer. References 7 and 8 are essential for working with the SDK-51, and Reference 9 is of great help.

Digital Control Equations: A procedure for developing digital control equations has been developed, which meets specific requirements:

* A given design damping value.
* Insensitivity to steady acceleration, including gravity.
* A given design centering stiffness.

* A specified phase margin.

Equations based on rectangular integration have been demonstrated. Improved equations, based on w-transform theory, have been developed, which show small changes from the demonstrated values. Finally, real damping vs. frequency has been calculated for both sets of equations, and results of these calculations have been presented in this report.

<u>Floating-Point Calculations</u>: The demonstrated equations used floating-point subroutines which were developed for the 8051 series microcontrollers.

<u>Pulse Width Modulation</u>: A pulse-width modulator (PWM) was developed for the proof-mass actuator. This draws no current and therefore develops no heat when the actuator is in a quiescent state.

<u>Word Length</u>: It is recognized that four factors determine the accuracy of the control program, they are:

* Possible loss of accuracy due to limited word length in input and output.
* Possible loss of significance due to overflow or underflow during internal calculations.
* Digital noise due to inadequate word length.
* Long computational time due to arithmetic complexity.

Experience with the Z80 and the current 8051 series control programs gave no indications of problems due to input or output word length. For example, when programmed as a pure spring, the proof-mass appears to behave smoothly, without any apparent 'stair-step' feel when operated manually. However, with the 16-bit Z80 system, there were definite indications of internal number overflow. Possibly, these could have been corrected by shifting to the middle 8 bits for input and output. However, the 8051 series is not well adapted to 16-bit arithmetic, and this is why the floating-point approach was tried. Several other schemes could have been used, overall, one might consider any of the following:

* Signed 7-bit arithmetic (8-bit total).
* Signed 15-bit arithmetic (16-bit total).
* Signed 15-bit arithmetic with shift (16-bit total).
* Signed 7-bit mantissa and exponent (16-bit total).
* Signed 11-bit mantissa and signed 3-bit exponent (16-bit total).
* Signed 15-bit mantissa and signed 7-bit exponent (24-bit total).

Since the 8031 chip was used, requiring two ports dedicated to memory access, the SDK-51 system was limited to an 8-bit A/D. Also, but for different reasons, the PWM was limited to 8 effective bits. Since no advantage was seen in going to more bits in either case, the extra hardware which would have been required did not have to be used.

<u>Future Development</u>: This report concludes work under the NASA grant, so that any future work will be carried out on internal funds. However, the development of a slave-master system is of

particular interest, because it will make it possible to change the gains on individual controllers, as might be required in operation. Presently available development systems make this a difficult task, because only a single 8051 can be simulated at any one time. Specifically, the proposed development would include the following:

* Installing a slave 8031 in the wire-wrap area of the SDK-51.
* Installing 2K of RAM so that it can be programmed from the SDK-51, but can be used to run programs on the slave.
* Provision for installation of a 2K EPROM in the RAM slot.
* Provision for programming the EPROM in place.
* Interconnection of the serial lines on the two 8031's.
* Use of the four high address bits on the slave 8031 to control A/D and other board functions.

This system would be used to develop slave controller programs on EPROM which would be used in building separate controller boards. The EPROM programming capability would also be used to develop additional library programs for the SDK-51.

CONCLUSIONS AND RECOMMENDATIONS

* The 8051 series microcontrollers are capable of controlling the proof-mass actuator.
* Eight-bit input and output appears adequate, however, with the availabilty of the additional ports on the 8751, A/D's and D/A's with more bits pose no problem and would require

little extra time.

* Although the floating-point arithemetic gave good results, other arithmetic schemes might require less computing time. The question requires more investigation than was given in the present work.

* The parallel realization design procedure described in this report worked well and appears to be adequate.

* The recommended design procedure requires a fair amount of calculation, especially if the phase margin is to be optimal. For best results, it might be advisable to write a computer program to determine parameters for the difference equations.

REFERENCES

1. Haviland, J.K., "Digital Control System for Space Structure Dampers," University of Virginia, Department of Mechanical and Aerospace Engineering, Proposal No. MAE-NASA-2548-83 to the NASA Langley Research Center, January 1983.

2. Pilkey, W.D., and Haviland, J.K., "Large Space Structure Damping Design," University of Virginia, Department of Mechanical and Aerospace Engineering, Report No. UVA/528201/MAE83/101. February 1983.

3. Haviland, J.K., "Digital Control System for Space Structural Dampers," University of Virginia, Department of Mechanical and Aerospace Engineering, Semi-Annual Report No. UVA/528224/MAE84/101, January 1984.

4. Haviland, J.K., "Digital Control System for Space Structural

Dampers," University of Virginia, Department of Mechanical and Aerospace Engineering, Report No. UVA/528224/MAE85/102, July 1984.

5. Katz, P, "Digital Control Using Microprocessors," Prentice/Hall, 1981.

6. Kuo, B.C., "Digital Control Systems," Holt, Rinehart and Winston, Inc, 1980.

7. "Microcontroller Handbook," INTEL Corporation, Order No. 210918-002, Intel Literature Department, 3065 Bowers Ave., Santa Clara, CA 95051, 1984.

8. "SDK-51 MCS-51 System Design Kit User's Guide," Order No. 121588-002, Intel Literature Department, 3065, Bowers Ave., Santa Clara, CA 95051, 1981.

9. Boyet, H, and Katz, R., "The 8051: Programming, Interfacing Applications," Microprocessor Training Publications Inc., New York.

APPENDIX A

EXPERIMENTAL PROGRAM FOR SDK-51 BOARD

This program was written to assist in the overall development of the wire-wrapped controller added to INTEL's SDK-51 development board. It is loaded from a cassette tape, titled ABC9, and responds to the keyboard command 'GO FROM 0', by executing a program called DEMO1. While executing this or any other program, it continuously polls the keyboard, and responds to any inputs with ASCII values of 20H to 5FH by a subroutine call to the appropriate location in a table. If it encounters RET, it simply returns to the current program, but, if it encounters JMP addr., it jumps to a new program. The following is a list of keyboard entries which cause jumps to new programs, the number in parenthesis is the address of the program:

C=COIL (0568H): The program waits for two hex characters in 2's complement form, which is output to the coil. Used to measure coil force output.

D=DISPLAY (01A8H): Displays four hex bytes, in 2's complement form, indicating readings of the four analog input ports. Used for calibration of analog inputs.

E=ENTER (0454H): Enters floating-point contents of 06,07 into first stack location, moves two stack contents up, and loses contents of third stack location.

F=FIX (04E0H): Fixed-point equivalent of floating-point number in 06,07 is stored in 06 (and displayed).

N=NEGATE (0448H): Floating-point contents of 06,07 are negated and replaced in 06,07 (and displayed).

P=P1-D Program (0300H): The P-Program, as described in the test, is run.

Q=continue P1-D Program (0308H): The P-Program is restarted with current parameters (i.e., default values are not read).

R=READ (0470H): First floating-point number on stack is read into 06,07 (and displayed). Remainder of stack is moved down, and third stack location is left unchanged.

T=T version of P1-D Program (0330H): The T-Program, as described in the text, is run.

U=continue T version (0338H): The T-Program is restarted with current parameters (i.e., default values are not read).

X=EXPONENT (0424H): The program waits for two hex characters, representing the exponent, and enters them into 07 (and displays them). This must follow the mantissa entry, which writes over the current exponent.

Z=NORMALIZE (043CH): The floating-point contents of 06,07 are normalized and replaced in 06,07 (and displayed).

Space Bar, Shift 0,1 (04D0HO: Parameters are entered from floating point numbers in 06,07, to be followed by U to restart T-Program, according to following table:

Space Bar ..... n

Shift 1 c

Shift 2 .... ω_N

Shift 2 to Shift 9 (04D0H): Parameters are entered from floating- point numbers in 06,07, to be followed by Q to restart P-Program, according to the following table:

Shift 3 .... I_n

Shift 4 .... T

Shift 5 .... ω_A

Shift 6 .... ω_P

Shift 7 .... G_A

Shift 8 .... G_P

Shift 9 .... G_V

'*'=MULTIPLY (0490H): Floating -point contents of first stack position are multiplied by contents of 06,07, and replaced in 06,07 (and displayed). Stack contents are moved down, so that both multiplier and multiplicand are lost.

'+'=ADD (04A0H): Floating-point contents of 06,07 are added to contents of first stack position, and replaced in 06,07 (and displayed). Stack contents are moved down, so that both addends are lost.

'-'=SUBTRACT (04B0H): Floating-point contents of 06,07 are subtracted from contents of first stack position, and replaced in 06,07 (and displayed). Stack contents are moved down, so that subtactor and subtrahend are lost.

'.'=MANTISSA (0418H): The program waits for two hex characters, and enters them in both 06 and 07. The contents of 06 will represent the mantissa, but the exponent should follow to be placed in 07.

0 to 3 (0186H): DEMO0 to DEMO3 are run, according to the following table:

0 .... DEMO0 places Channel #0 input at output.
1 .... DEMO1 places Channel #1 input at output.
2 .... DEMO2 places Channel #2 input at output.
3 .... DEMO3 places Channel #3 input at output.

From the point-of-view of proof-mass controller development, the most important items are the two versions of the P1-D controller, referred to as as the P- and T- Programs. These use floating-point subroutines, and make use of the timer interrupt feature of the 8031. A key to internal data memory and a listing of program ABC9 follows.

Key to Internal Data Memory

Address	Function
00,01	R0 and R1 pointers
02	R2 is used for display
03	R3 cycle counter
04	R4 exponent
05	R5 shift counter
06,07	R6, R7 floating point results
20	Bit 00 = sign
	Bit 01 = flag
22	Channel counter
23	Display counter
24	Key input
25	n = # cycles
26	Present output, 2's complement hex
27	Output during next calculation cycle
2A,2B	n (default value)
2C,2D	c (default value)
2E,2F	ω_N (default value)
30,31	Calculator stack #1
32,33	Calculator stack #2
34,35	Calculator stack #3
36	I_n (default value)
38,39	T (default value)
3A,3B	ω_A (default value)
3C,3D	ω_P (default value)
3E,3F	G_A (default value)

40,41	G_P (default value)
42,43	G_V (default value)
44,45	$\omega_A T$
46,47	$\omega_P T$
48,49	$G_A T$
4A,4B	$G_P T$
50,51	$-x_P$
52,53	x_A
54,55	$-x_L$
56,57	x_S
58,59	u_V
5A,5B	u_P
5C,5D	u_A

```
0000=AJMP 0180        RESET - Jump to DEMO 9
0002=NOP
0003=LJMP E003        INTERRUPT I - Required for SDK-51
0006=NOP
0007=NOP
0008=NOP
0009=NOP
000A=NOP              TIMER 0 INTERRUPT
000B=CLR 8E           Turn TIMER I off
000D=CPL B5           Invert R.H. Voltage on Coil
000F=CLR 01           Clear Flag
0011=ACAL 0027        Call Subroutine
0013=RETI             Return from Interrupt
0014=NOP
0015=NOP
0016=NOP
0017=NOP
0018=NOP
0019=NOP
001A=NOP              TIMER I INTERRUPT
001B=CLR 8E           Turn Timer I off
001D=CPL B5           Invert R.H. Voltage on coil
001F=RETI             Return from Interrupt
0020=NOP
0021=NOP
0022=NOP
0023=NOP
0024=NOP
0025=NOP
0026=RETI             TIMER 0 SUBROUTINE
0027=PUSH D0          Save PSW
0029=PUSH E0          Save A
002B=MOV A,26         Output to A
002D=SETB C           Set Carry
002E=RLC A            Rotate output left
002F=JC 0032          Skip next instruction if negative
0031=CPL A            Complement output
0032=MOV 8C,A         Set TIMER I
0034=MOV B5,C         RH Voltage high if output negative
0036=CPL C            Invert sign
0037=MOV B4,C         L.H. voltage low if output negative
0039=SETB 8E          Start TIMER I
003B=DJNZ R3,0047     Skip 5 instructions if R3 not zero
003D=MOV R3,25        Reset
003F=MOV A,27         Update
0041=MOV 26,A             output
0043=SETB 01          Set flag
0045=SETB B0          Set Oscilloscope Signal
0047=POP E0           Retrieve A
0049=POP D0           Retrieve RSW
004B=RET              Return
```

```
                                SUBROUTINE to READ A/D
0054=MOV C,10                   Low digit
0056=MOV 90,C                      to pin 1.0
0058=MOV C,11                   Next digit
005A=MOV 91,C                      to pin 1.1
005C=SETB 93                    Enable A/D
005E=CLR 92                     Trigger
0060=SETB 92                       A/D
0062=NOP                        Wait
0063=NOP                        Wait
0064=MOV 90,#FF                 Set Port 3 to read
0067=CLR B3                     Set Transceiver to read
0069=MOV A,90                   Read A/D into A
006B=CLR 93                     Disable A/D
006D=SETB B3                    Set transceiver to write
006F=MOV R4,#00                 Set exponent to zero
0071=RET                        Return

                                SUBROUTINE FOR INITIAL SETUP
007B=MOV 88,#00                 Set TCON = 0
007E=MOV 89,#23                 Set TMOD.  TIMER 0 = Mode 3, TIMER 1 = Mode 2
0081=MOV 97,#00                 Set PCON = 0
0084=MOV 98,#00                 Set SCON = 0
0087=MOV A8,#EB                 Set IE.  Enable both timer interrupts
008A=MOV B8,#08                 Set IP.  Timer 1 has priority
008D=RET                        Return

                                SUBROUTINE TO POLL KEYBOARD
0094=CLR C                      Clear carry
0095=LCAL E00C                  Look for keyboard entry
0098=JNC 00A3                   Jump to return on no entry
009A=LCAL E009                  Read ASC II input
009D=ANL A,#7F                  Set bit #7 to zero
009F=MOV 24,A                   Save key input
00A1=ACAL 00AD                  CALL INTERPRET
00A3=RET                        Return

                                SUBROUTINE TO INTERPRET KEYSTROKES
00AD=ANL A,#70                  Remove 4 low bits
00AF=CJNE A,#20,00B4            Test for 20H to 2FH
00B2=SJMP 00C1                  Jump if successful
00B4=CJNE A,#30,00B9            Test for 30H to 3FH
00B7=SJMP 00C1                  Jump if successful
00B9=CJNE A,#40,00BE            Test for 40H to 4FH
00BC=SJMP 00C1                  Jump if successful
00BE=CJNE A,#50,00CC            Test for 50H to 5FH
00C1=MOV A,24                   Get original entry
00C3=NOP                        NOP
00C4=ANL A,#3F                  Skip two high bits
00C6=RL A                       Multiply by 2
00C7=MOV DPTR,#0100             Set DATA POINTER to start of table
00CA=ACAL 00CD                  Make it a subroutine call
00CC=RET                        Return from subroutine
00CD=CLR 8C                     Stop TIMER 0
00CF=CLR B4                     L.H. voltage to zero
00D1=CLR B5                     R.H. voltage to zero
00D3=JMP @A+DPTR                Jump to table
```

SUBROUTINE TO DISPLAY & WAIT

```
00DA=LCAL E00F        Clear display
00DD=MOV R2,#2E       Output
00DF=LCAL E006           period
00E2=MOV R2,06        Output
00E4=LCAL E015           mantissa
00E7=MOV R2,#58       Output
00E9=LCAL E006           Cap. X
00EC=MOV R2,07        Output
00EE=LCAL E015           exponent
00F1=MOV R2,#20       Output
00F3=LCAL E006           space
00F6=MOV R2,24        Output
00F8=LCAL E006           keystroke
00FB=ACAL 0094        Look for new
00FD=SJMP 00FB           keystroke
```

TABLE = KEYSTROKES 40H to 57H

```
0100=RET
0101=RET
0102=RET
0103=RET
0104=RET
0105=RET
0106=AJMP 0568        C = Coil Force
0108=AJMP 01A8        D = Display four analog inputs
010A=AJMP 0454        E = Enter onto stack
010C=AJMP 04E0        F = Fixed Decimal
010E=RET
010F=RET
0110=RET
0111=RET
0112=RET
0113=RET
0114=RET
0115=RET
0116=RET
0117=RET
0118=RET
0119=RET
011A=RET
011B=RET
011C=AJMP 0448        N = Negate
011E=RET
011F=RET
0120=AJMP 0300        P = PI-D Program
0122=AJMP 0308        Q = Continue P
0124=AJMP 0470        R = Read stack
0126=RET
0127=RET
0128=AJMP 0330        T = Alternate PI-D Program
012A=AJMP 0338        U = Continue T
012C=RET
012D=RET
012E=RET
012F=RET
```

```
                          TABLE:  KEYSTROKES 58H to 5FH; 20H to 3FH
0130=AJMP 0424            X = Exponent
0132=RET
0133=RET
0134=AJMP 043C            Z = Normalize
0136=RET
0137=RET
0138=RET
0139=RET
013A=RET
013B=RET
013C=RET
013D=RET
013E=RET
013F=RET
0140=NOP
0141=NOP
0142=NOP
0143=NOP
0144=AJMP 04C0            Shift 0 to 2
0146=NOP
0147=NOP
0148=NOP
0149=NOP
014A=NOP
014B=NOP
014C=NOP
014D=NOP
014E=NOP
014F=NOP
0150=NOP
0151=NOP
0152=AJMP 04D0            Shift 3 to 9
0154=AJMP 0490            '*' = Multiply
0156=AJMP 04A0            '+' = Add
0158=RET
0159=RET
015A=AJMP 04B0            '-' = Subtract
015C=AJMP 0418            '.' = Mantisser
015E=RET
015F=RET
0160=AJMP 0186            DEMO 1 . Proximeter test
0162=AJMP 0186            DEMO 2 . Accelerometer test
0164=AJMP 0186            DEMO 3 . LVDT test
0166=AJMP 0186            DEMO 4 . Signal generator test
0168=RET
    ↓
017F=RET
```

DEMO 0-3 PROGRAMS

```
0180=LCAL E00C          Read keyboard
0183=MOV 24,#30         Set input channel to #0
0186=MOV 81,#60         Stack pointer = 60
0189=MOV 25,#01         Counter input = 0
018C=ACAL 007B          CALL INITIAL SETUP
018E=SETB 8C            Start TIMER 0
0190=CLR 01             Clear flag
0192=MOV A,24           Keystroke (0 to 3)
0194=MOV 22,A                into Channel #
0196=ACAL 0054          Call  READ A/D
0198=MOV 27,A           A/D input to 27
019A=CLR B0             Clear oscilloscope signal
019C=ACAL 0094          CALL POLL KEYBOARD
019E=JB 01,0190         Jump if flag high
01A1=SJMP 019E          Wait for interrupt
```

DISPLAY 4 INPUTS

```
01A8=MOV 81,#60         Stack pointer = 60
01AB=NOP
01AC=NOP
01AD=NOP
01AE=ACAL 0078          CALL INITIAL SETUP
01B0=MOV A8,#E1         Reset IE - Disable interrupts
01B3=MOV B8,#00         Reset IP - Cancel interrupt priorities
01B6=SETB 8C            Start TIMER 0
01B8=ACAL 01C0          CALL INPUT subroutine
01BA=ACAL 0094          CALL POLL KEYBOARD
01BC=SJMP 01B8          Continue
```

SUBROUTINE FOR INPUT

```
01C0=JNB 8D,01E1        Return if TIMER 0 flag low
01C3=CLR 8D             Clear TIMER 0 overflow flag
01C5=INC 23             Increment display counter
01C7=MOV A,23           Return on
01C9=JNZ 01E1              nonzero display counter
01CB=LCAL E00F          Clear display
01CE=MOV R0,#22         Set channel #0
01D0=MOV @R0,#00              to zero
01D2=ACAL 0054          CALL READ A/D
01D4=MOV R2,A           Display
01D5=LCAL E015            reading
01D8=MOV R2,#2C         Output
01DA=LCAL E006            comma
01DD=INC @R0            Increment channel #
01DE=CJNE @R0,#04,01D2  Continue if channel # not 5
01E1=RET                Return
```

	SUBROUTINE TO CORRECT MANTISSA OVERFLOW
01F0=JNB D2,01F7	Return on no OVERFLOW
01F3=RRC A	Rotate right
01F4=CJNE R4,#7F,01F8	Jump if exponent not maximum
01F7=RET	Return
01F8=INC R4	Increment exponent
01F9=RET	Return

	SUBROUTINE TO NEGATE A, 04
0204=CPL A	Complement A
0205=ADD A,#01	Add Unity
0207=ACAL 01F0	Correct overflow
0209=RET	Return

	SUBROUTINE TO NORMALIZE MANTISSA
0210=JB E7,0219	Jump if negative
0213=JB E6,0221	Return if normalized
0216=SETB C	Set carry if positive to enter 1's
0217=SJMP 021D	
0219=JNB E6,0221	Return if normalized
021C=CLR C	Clear carry if negative to enter 0's
021D=RLC A	Rotate left through carry
021E=CJNE R4,#80,0222	Jump if exponent not minimum
0221=RET	Return
0222=DEC R4	Decrement exponent
0223=SJMP 0210	Continue

	SUBROUTINE TO SHIFT MANTISSA TO RIGHT
0228=INC R5	Increment shift counter
0229=DJNZ R5,022C	Decrement shift counter, jump if nonzero
022B=RET	Return
022C=MOV C,E7	Set carry = sign bit
022E=RRC A	Rotate, right
022F=SJMP 0229	Continue

	SUBROUTINE MOVE A, 04 TO 06, 07
0234=MOV R6,A	A to 06
0235=MOV R7,04	04 to 07
0237=RET	Return

```
                                  SUBROUTINE @RO + @R1 → 06,07
0240=INC R0                       Increment R0 to exponent address
0241=INC R1                       Increment RI to exponent address
0242=MOV A,@R0                    Exponent = #0
0243=CLR C                        Clear carry
0244=SUBB A,@R1                   Subtract exponent #0
0245=CPL C                        Complement carry
0246=JNB D2,024A                  Skip next instruction if no overflow
0249=RRC A                        Rotate right
024A=JB E7,0258                   Skip eight instructions if negative
024D=MOV R5,A                     Set exponent difference in shift counter
024E=MOV 04,@R0                   Store exponent #0 in 04
0250=DEC R1                       Decrement RI to Mantissa address
0251=MOV A,@R1                    Mantissa #I to A
0252=ACAL 0228                    CALL SHIFT MANTISSA
0254=DEC R0                       Decrement R0 to Mantissa address
0255=ADD A,@R0                    Add Mantissa #0
0256=SJMP 0263                    Jump to exit
0258=CPL A                        Complement to get exponent difference
0259=INC A                        Add I to get 2's complement
025A=MOV R5,A                     Set exponent difference in shift counter
025B=MOV 04,@R1                   Store exponent #I in 04
025D=DEC R0                       Decrement R0 to Mantissa address
025E=MOV A,@R0                    Mantissa #0 to A
025F=ACAL 0228                    CALL SHIFT MANTISSA
0261=DEC R1                       Decrement RI to Mantissa address
0262=ADD A,@R1                    Add Mantissa #I
0263=ACAL 01F0                    (Exit) CALL CORRECT MANTISSA
0265=ACAL 0210                    CALL NORMALIZE MANTISSA
0267=ACAL 0234                    CALL MOVE A, 04 to 06, 07
0269=RET                          Return

                                  SUBROUTINE NEGATE @ RI
026C=MOV A,@R1                    Mantissa #I to A
026D=INC R1                       Increment RI to exponent address
026E=MOV 04,@R1                   Exponent #I to 04
0270=DEC R1                       Decrement RI to Mantissa address
0271=ACAL 0204                    CALL NEGATE A, 04
0273=ACAL 0210                    CALL NORMALIZE MANTISSA
0275=MOV @R1,A                    Store Mantissa
0276=INC R1                       Increment RI to exponent address
0277=MOV @R1,04                   Store exponent
0279=DEC R1                       Restore RI
027A=RET                          Return

                                  SUBROUTINE @ R0 - @RI → 06, 07
027C=ACAL 026C                    CALL NEGATE @ RI
027E=ACAL 0240                    CALL @ R0 + @R1 → 06, 07
0280=CJNE R1,#06,0285             Skip next instruction if R0 not 06
0283=SJMP 0287                    Jump to return
0285=ACAL 026C                    CALL NEGATE @ RI
0287=RET                          Return
```

```
02D0=CLR A           SUBROUTINE FLOAT @R0 to @R1
                     Clear A
02D1=MOV 04,A        Zero to 04
02D3=MOV A,@R0       Mantissa #0 to A
02D4=ACAL 0210       CALL NORMALIZE MANTISSA (ENTRY)*
02D6=MOV @R1,A       A to Mantissa #1
02D7=INC R1          Increment R1 to exponent
02D8=MOV @R1,04      04 to exponent #1
02DA=DEC R1          Restore R1
02DB=RET             Return
```

*ENTRY FOR FLOAT A, 04 to @R1

```
                     SUBROUTINE FIX @R0 to @R1
02DC=INC R0          Increment R0 to exponent
02DD=MOV A,@R0       Exponent #0 to A
02DE=DEC R0          Decrement R0 to Mantissa
02DF=JB E7,02EB      Skip 5 instructions if negative
02E2=JZ 02ED         Skip 6 instructions if zero
02E4=MOV A,@R0       Mantissa #0 to A
02E5=ANL A,#80       Keep sign of Mantissa
02E7=ACAL 0210       CALL NORMALIZE MANTISSA
02E9=SJMP 02F1       Jump to exit
02EB=CPL A           Complement negative exponent
02EC=INC A                2's complement
02ED=MOV R5,A        Set shift counter
02EE=MOV A,@R0       Mantissa #0 to A
02EF=ACAL 0228       CALL SHIFT MANTISSA
02F1=MOV @R1,A       (Exit) A to Mantissa #1
02F2=RET             Return
```

```
                     SUBROUTINE STORE 06, 07 in @R1
02F4=MOV @R1,06      06 to Mantissa #1
02F6=INC R1          Increment R1 to exponent
02F7=MOV @R1,07      07 to exponent #1
02F9=DEC R1          Restore R1
02FA=RET             Return
```

P-PROGRAM

```
0300=MOV 81,#60          Set stack pointer to 60
0303=MOV DPTR,#04F0      Set data pointer to TABLE 1
0306=ACAL 0356           CALL READ P PARAMETERS
0308=ACAL 0362           CALL MULTIPLY BY T
030A=ACAL 0380           CALL READ INPUTS
030C=ACAL 03A0           CALL CALCULATE Up
030E=ACAL 03D8           CALL CALCULATE Ua
0310=MOV R0,#56          SET R0 to Xs
0312=MOV R1,#5A          SET R1 to Up
0314=ACAL 0240           CALL @R0 + @R1 to 06, 07
0316=MOV R0,#06          SET R0 to 06
0318=MOV R1,#5C          SET R1 to Ua
031A=ACAL 0240           CALL @ R0 + @R1 to 06, 07
031C=MOV R0,#06          SET R0 to 06
031E=MOV R1,#27          SET R1 to OUTPUT
0320=ACAL 02DC           CALL FIX  @R0 to @R1
0322=CLR B0              CLEAR oscilloscope signal
0324=ACAL 0094           CALL POLL KEYBOARD
0326=JB 01,030A          LOOP if flag high
0329=SJMP 0326           Wait for interrupt
```

T-PROGRAM

```
0330=MOV 81,#60          Set stack pointer to 60
0333=MOV DPTR,#05F8      Set data pointer to TABLE 2
0336=ACAL 0340           CALL READ T PARAMETERS
0338=ACAL 0500           CALL CALCULATE PARAMETERS
033A=AJMP 0308           Jump to P-Program
```

SUBROUTINE READ T PARAMETERS

```
0340=ACAL 007B           CALL INITIAL SETUP
0342=MOV R1,#2A          Set R1 to n
0344=MOVX A,@DPT         TABLE 2 to A
0345=MOV @R1,A           Store A
0346=INC DPTR            Increment data pointer
0347=INC R1              Increment R1
0348=CJNE R1,#30,0344    LOOP until R1 = 30
034B=RET                 Return
```

SUBROUTINE READ P PARAMETERS

```
0356=ACAL 007B           CALL INITIAL SETUP
0358=MOV R1,#36          Set R1 to In
035A=MOVX A,@DPT         TABLE 1 to A
035B=MOV @R1,A           Store A
035C=INC DPTR            Increment data pointer
035D=INC R1              Increment R1
035E=CJNE R1,#44,035A    LOOP until R1 = 44
0361=RET                 Return
```

SUBROUTINE SHIFT @ R0 → @R1

```
0288=MOV A,@R0          Mantissa #0 to A
0289=MOV @R1,A          A to Mantissa #1
028A=INC R0             Increment R0 to exponent address
028B=INC R1             Increment R1 to exponent address
028C=MOV A,@R0          Exponent #0 to A
028D=MOV @R1,A          A to exponent #1
028E=DEC R0             Restore R0
028F=DEC R1             Restore R1
0290=RET                Return
```

SUBROUTINE @R0 * @R1 → 06, 07

```
0294=CLR 00             Clear sign flag
0296=INC R0             Increment R0 to exponent address
0297=INC R1             Increment R1 to exponent address
0298=MOV A,@R0          Exponent #0 to A
0299=ADD A,@R1          Add exponent #1
029A=JNB D2,02A5        Skip 4 instructions if no overflow
029D=JNC 02A3           Skip 2 instructions if positive
029F=MOV A,#80          Set exponent to 80H
02A1=SJMP 02A5          Skip next instruction
02A3=MOV A,#7F          Set exponent to 7FH
02A5=MOV R4,A           Exponent to 04
02A6=DEC R0             Decrement R0 to Mantissa address
02A7=DEC R1             Decrement R1 to Mantissa address
02A8=MOV A,@R0          Mantissa #0 to A
02A9=JNB E7,02B0        Skip two instructions if positive
02AC=CPL 00             Complement sign bit
02AE=ACAL 0204          CALL NEGATE A, 04
02B0=NOP                NOP
02B1=MOV F0,A           Mantissa #0 to B
02B3=MOV A,@R1          Mantissa #1 to A
02B4=JNB E7,02BB        Skip two instructions if positive
02B7=CPL 00             Complement sign bit
02B9=ACAL 0204          CALL NEGATE A, 04
02BB=CLR C              Clear carry
02BC=RLC A              Rotate left
02BD=MUL AB             Multiphy A * B
02BE=MOV A,F0           Product to A
02C0=NOP                NOP's
02C1=NOP                 to
02C2=NOP                 be
02C3=NOP                 removed
02C4=NOP                -
02C5=NOP                -
02C6=JNB 00,02CB        Skip two instructions if sign positive
02C9=ACAL 0204          NEGATE A, 04
02CB=ACAL 0210          CALL NORMALIZE MANTISSA
02CD=ACAL 0234          CALL MOVE A, 04 to 06, 07
02CF=RET                Return
```

SUBROUTINE MULTIPLY BY T

```
0362=MOV R0,#3A        Set R0 to ωa
0364=MOV R1,#44        Set R1 to ωat
0366=PUSH 01           Save R1
0368=MOV R1,#38        Set R1 to T
036A=ACAL 0294         CALL @R0 * @R1 to 06, 07
036C=POP 01            Retrieve R1
036E=ACAL 02F4         CALL STORE 06, 07 in @R1
0370=INC R0            Increment R0
0371=INC R0                twice
0372=INC R1            Increment R1
0373=INC R1                twice
0374=CJNE R1,#4C,0366  Loop until R1 = 4C
0377=MOV A,36          STORE In
0379=MOV 25,A              in 25
037B=SETB 8C           Start TIMER 0
037D=RET               Return
```

SUBROUTINE READ INPUTS

```
0380=CLR 01            Clear flag
0382=MOV 22,#00        Set channel # to 0
0385=MOV R1,#50        Set R1 to - Xp
0387=ACAL 0054         CALL READ A/D
0389=ACAL 02D4         CALL FLOAT A, 04 to @R1
038B=INC 22            Increment channel #
038D=INC R1            Increment R1
038E=INC R1                twice
038F=CJNE R1,#58,0387  Loop until R1 = 58
0392=MOV R0,#50        Set R0 to - Xp
0394=MOV R1,#54        Set R1 to -Xl
0396=ACAL 0240         CALL @R0 + @R1 to 06, 07
0398=ACAL 02F4         CALL STORE 06, 07 in @ R1
039A=RET               Return
```

SUBROUTINE CALCULATE Up

```
03A0=MOV R0,#46        Set R0 to ωpt
03A2=MOV R1,#5A        Set R1 to Up
03A4=ACAL 0294         ωpTup to 06, 07
03A6=MOV R0,#06        Set R0 to 06, 07
03A8=MOV R1,#58        Set R1 to Uv
03AA=ACAL 027C         ωpTup - Uv to 06, 07
03AC=ACAL 02F4         ωpTup - Uv to Uv
03AE=MOV R0,#4A        R0 set to Gpt
03B0=MOV R1,#54        R1 set to Xp
03B2=ACAL 0294         -XpGpT to 06, 07
03B4=MOV R1,#58        Set R1 to Uv
03B6=MOV R0,#06        Set R0 to 06, 07
03B8=ACAL 027C         Updated Uv to 06, 07
03BA=ACAL 02F4         Updated Uv to Uv
03BC=MOV R0,#42        Set R0 to Gv
03BE=MOV R1,#54        Set R1 to -Xp
03C0=ACAL 0294         -XpGv to 06, 07
03C2=MOV R0,#58        Set R0 to Uv
03C4=MOV R1,#06        Set R1 to 06
03C6=ACAL 0240         Uv - XpGv to 06, 07
03C8=MOV R1,#5A        Set R1 to Up
03CA=ACAL 02F4         Uv - XpGv to Up
03CC=RET               Return
```

```
                          SUBROUTINE CALCULATE Ua
03D8=MOV R0,#44           Set R0 to ωaT
03DA=MOV R1,#5C           Set R1 to Ua
03DC=ACAL 0294            ωaTUa to 06, 07
03DE=MOV R0,#06           Set R0 to 06
03E0=MOV R1,#5C           Set R1 to Ua
03E2=ACAL 027C              ωaTUa - Ua to 06, 07
03E4=ACAL 02F4              ωaTUa - Ua to Va
03E6=MOV R0,#48           Set R0 to GaT
03E8=MOV R1,#52           Set R1 to Xa
03EA=ACAL 0294              GaTXa to 06, 07
03EC=MOV R1,#5C           Set R1 to Ua
03EE=MOV R0,#06           Set R0 to 06, 07
03F0=ACAL 027C              Updated Ua to 06,07
03F2=ACAL 02F4              Updated Ua to Ua
03F4=RET                  Return

                          SUBROUTINE READ HEX BYTE
0400=LCAL E009            CALL READ KEY
0403=MOV R2,A             Store in R2
0404=LCAL E01B            CALL CONVERT TO HEX
0407=SWAP A               Place in top 4 bits
0408=MOV R7,A             Store in R7
0409=LCAL E009            CALL READ KEY
040C=MOV R2,A             Store in R2
040D=LCAL E01B            CALL CONVERT TO HEX
0410=ADD A,R7             Add to R7
0411=MOV R7,A             Store in R7
0412=RET                  Return

                          PROGRAM '.' TO READ MANTISSA
0418=MOV 81,#60           Set stack pointer to 60
041B=ACAL 0400            CALL READ HEX BYTE
041D=MOV R6,07            Store in 06
041F=AJMP 00DA            Jump to DISPLAY AND WAIT

                          PROGRAM 'X' To READ EXPONENT
0424=MOV 81,#60           Set stack pointer to 60
0427=ACAL 0400            CALL READ HEX TYTE
0429=AJMP 00DA            Jump to DISPLAY AND WAIT

                          SUBROUTINE TO NORMALIZE
0430=MOV A,R6             06 to A
0431=MOV 04,R7            07 to 04
0433=ACAL 0210            CALL TO NORMALIZE A, 04
0435=MOV R6,A             A to 06
0436=MOV R7,04            04 to 07
0438=RET                  Return

                          PROGRAM 'Z' NORMALIZE DISPLAY
043C=MOV 81,#60           Set stack pointer to 60
043F=ACAL 0430            CALL NORMALIZE
0441=AJMP 00DA            Jump to DISPLAY AND WAIT
```

```
                              PROGRAM 'N' to NEGATE DISPLAY
0448=MOV 81,#60               Set stack pointer to 60
044B=MOV R1,#06               Set R1 to 06
044D=ACAL 026C                CALL NEGATE @R1
044F=AJMP 00DA                Jump to DISPLAY and WAIT

                              PROGRAM 'E' to ENTER STACK
0454=MOV 81,#60               Set stack pointer to 60
0457=MOV R1,#35               Set R1 to STACK 3
0459=MOV R0,#33               Set R2 to STACK 2
045B=MOV A,@R0                Old stack to A
045C=MOV @R1,A                A to new stack
045D=DEC R0                   Decrement R0
045E=DEC R1                   Decrement R1
045F=CJNE R0,#2F,045B         Loop until R0 = 2FH
0462=MOV @R1,07               07 to stack. 1 exponent
0464=DEC R1                   Decrement R1
0465=MOV @R1,06               06 to Stack 1 Mantissa
0467=AJMP 00DA                Jume to DISPLAY and WAIT

                              PROGRAM 'R' to READ STACK
0470=MOV 81,#60               Set stack pointer to 60
0473=MOV R0,#30               Set R0 to STACK 1
0475=MOV R1,#06               Set R1 to 06
0477=ACAL 0288                CALL @R0 to @R1
0479=ACAL 0480                CALL SHIFT STACK
047B=AJMP 00DA                Jump to DISPLAY and WAIT

                              SUBROUTINE TO SHIFT STACK
0480=MOV R0,#32               Set R0 to STACK 2
0482=MOV R1,#30               Set R1 to STACK 1
0484=MOV A,@R0                Old stack to A
0485=MOV @R1,A                A to new stack
0486=INC R0                   Increment R0
0487=INC R1                   Increment R1
0488=CJNE R0,#36,0484         Loop until R0 = 36
048B=RET                      Return

                              PROGRAM '*' to MULTIPLY
0490=MOV 81,#60               Set stack pointer to 60
0493=MOV R0,#30               Set R0 to STACK 1
0495=MOV R1,#06               Set R1 to 0G
0497=ACAL 0294                CALL @R0 * @R1 to 06, 07
0499=ACAL 0480                CALL SHIFT STACK
049B=AJMP 00DA                Jump to DISPLAY and WAIT
```

PROGRAM '+' to ADD

```
04A0=MOV 81,#60      Set stack pointer to 60
04A3=MOV R0,#30      Set R0 to STACK 1
04A5=MOV R1,#06      Set R1 to 06
04A7=ACAL 0240       CALL @R0 + @R1 to 06, 07
04A9=ACAL 0480       CALL SHIFT STACK
04AB=AJMP 00DA       Jump to DISPLAY and WAIT
```

PROGRAM '-' to SUBTRACT

```
04B0=MOV 81,#60      Set stack pointer to 60
04B3=MOV R0,#30      Set R0 to STACK 1
04B5=MOV R1,#06      Set R1 to 06
04B7=ACAL 027C       CALL @R0 - @R1 to 06, 07
04B9=ACAL 0480       CALL SHIFT STACK
04BB=AJMP 00DA       Jump to DISPLAY and WAIT
```

PROGRAM S.B., SHIFTS 1, 1. INPUTS

```
04C0=MOV 81,#60      Set stack pointer to 60
04C3=MOV A,24        Key input to A
04C5=CLR C           Clear carry
04C6=RLC A           Rotate A left
04C7=ADD A,#EA       Subtract 6
04C9=MOV R1,A        Result to R1
04CA=ACAL 02F4       CALL 06, 07 to @R1
04CC=AJMP 00DA       Jump to DISPLAY and WAIT
```

PROGRAM SHIFTS 1-9. INPUTS

```
04D0=MOV 81,#60      Set stack pointer to 60
04D3=MOV A,24        Key input to A
04D5=CLR C           Clear carry
04D6=RLC A           Rotate A left
04D7=ADD A,#F0       Subtract 16
04D9=MOV R1,A        Result to R1
04DA=ACAL 02F4       CALL 06, 07 to @R1
04DC=AJMP 00DA       Jump to DISPLAY and WAIT
```

PROGRAM 'F' to FIX DISPLAY

```
04E0=MOV 81,#60      Set stack pointer to 60
04E3=MOV R0,#06      Set R0 to 06
04E5=MOV R1,#06      Set R1 to 06
04E7=ACAL 02DC       CALL FIX @R0 to @R1
04E9=AJMP 00DA       Jump to DISPLAY and WAIT
```

	TABLE 1. PARAMETERS FOR PROGRAM P
CBYT 04F0=10,00,40,F9	In , T
CBYT 04F4=48,06,5E,07	ωa , ωp
CBYT 04F8=66,06,5E,05	Ga , Gp
CBYT 04FC=40,03,00,00	Gv

	SUBROUTINE - CALCULATE T PARAMETERS
0500=MOV DPTR,#05E8	SET data pointer to TABLE 2
0503=MOV R0,#06	SET R0 to 06
0505=ACAL 0558	CALL GET DATA
0507=MOV R1,#2A	SET R1 to n
0509=ACAL 0294	CALL @R0 + @R1 to 06, 07
050B=MOV R1,#36	SET R1 to In
050D=ACAL 02DC	CALL FIX @R0 to @R1
050F=ACAL 0558	CALL GET DATA
0511=MOV R1,#2A	SET R1 to n
0513=ACAL 0294	CALL @R0 * @R1 to 06, 07
0515=MOV R1,#38	SET R1 to T
0517=ACAL 02F4	CALL 06, 07 to @R1
0519=ACAL 0558	CALL GET DATA
051B=MOV R1,#2C	SET R1 to C
051D=ACAL 0294	CALL @R0 * @R1 to 06, 07
051F=MOV R1,#3A	SET R1 to ωa
0521=ACAL 02F4	CALL 06, 07 to @R1
0523=ACAL 0558	CALL GET DATA
0525=MOV R1,#2E	Set R1 to ωn
0527=ACAL 0294	CALL @R0 * @R1 to 06, 07
0529=MOV R1,#3C	SET R1 to ωp
052B=ACAL 02F4	CALL 06, 07 to @R1
052D=ACAL 0558	CALL GET DATA
052F=MOV R1,#2C	Set R1 to C
0531=ACAL 0294	CALL @R0 * @R1 to 06, 07
0533=MOV R1,#3E	SET R1 to Ga
0535=ACAL 02F4	CALL 06, 07 to @R1
0537=ACAL 0558	CALL GET DATA
0539=MOV R1,#2E	Set R1 to ωn
053B=ACAL 0294	CALL @R0 * @R1 to 06, 07
053D=MOV R1,#2E	Set R1 to ωn
053F=ACAL 0294	CALL @R0 * @R1 to 06, 07
0541=MOV R1,#42	SET R1 to Gv
0543=ACAL 02F4	CALL 06, 07 to @R1
0545=ACAL 0558	CALL GET DATA
0547=ACAL 0294	CALL @R0 * @R1 to 06, 07
0549=MOV R1,#2E	Set R1 to ωn
054B=ACAL 0294	CALL @R0 * @R1 to 06, 07
054D=MOV R1,#40	SET R1 to Gp
054F=ACAL 02F4	CALL 06, 07 to @R1
0551=RET	Return

SUBROUTINE TO GET DATA

```
0558=MOVX A,@DPT          Move from TABLE to A
0559=MOV 06,A             A to 06 (Mantissa)
055B=INC DPTR             Increment data pointer
055C=MOVX A,@DPT          Move from TABLE to A
055D=MOV 07,A             A to 07 (exponent)
055F=INC DPTR             Increment data pointer
0560=RET                  Return
```

PROGRAM 'C' FOR COIL FORCE

```
0568=MOV 81,#60           SET stack pointer to 60
056B=MOV 25,#01           SET 1 cycle
056E=ACAL 007B            CALL SETUP
0570=ACAL 0400            CALL READ HEX BYTE
0572=SETB 8C              Start TIMER 0
0574=CLR 01               Clear flag
0576=MOV A,07             07 (HEX BYTE) to A
0578=MOV 27,A             A to output
057A=CLR B0               Clear oscilloscope signal
057C=ACAL 0094            CALL POLL KEYBOARD
057E=JB 01,0574           Loop if flag high
0581=SJMP 057E            Wait for interrupt
```

TABLE 2 FOR T-PROGRAM

```
CBYT 05E8=40,FA,43,F5     2-7, 256x10-6
CBYT 05EC=73,02,40,04     3.60, 8
CBYT 05F0=51,03,75,FB     5.08, .0287
CBYT 05F4=40,00,00,00     .5, —
CBYT 05F8=40,05,50,04     n,c
CBYT 05FC=5E,04,00,00     ωn  —
```

APPENDIX B

SCHEMATICS

Six logic diagrams for the digital controller follow. Each has a sheet number, used when referring to connections between different sheets. Power and ground connections to standard DIPs are not shown, nor are despiking capacitors. The first five sheets refer to circuits on the wire-wrap area of the SDK-51 board, while sheet 6 refers to the PWM board.

Sheet 1, Figure B1: This shows the transceiver which was used to permit sharing of some pins between input and output functions. Also, the drivers for the PWM board are shown. These were needed because the signals from the 8031 where inadequate to drive the LED's in the opto-transistors.

Sheet 2, Figure B2: This shows the A/D converter, which is made up from a DAC0800 8-bit D/A converter, and a DM2502 successive approximation register. A high speed LM361 comparator is used to produce a TTL signal to the DM2502, which requires ten cycles to convergence. Presently, a 3 MHz clock is being used, so that convergence time is 3.33 microseconds. It is only achieving about 7-bit accuracy, but it is hoped that this will be improved with further adjustment.

Sheet 3, Figure B3: This shows the clock used to drive the A/D converter. It is derived from the 12 MHz crystal on the SDK-51 board, and provides four options, ranging from 6 MHz to 0.75 MHz, selectable with a jumper. Also, there is a one-and-one-only circuit to synchronize the start of the A/D with the clock.

Sheet 4, Figure B4: This shows the analog switch and demultiplexer circuit used to select the analog channel which is to be read by the A/D converter. A high speed operational amplifier is necessary to provide adequate output impedance combined with the switching speed required.

Sheet 5, Figure B5: This shows a typical analog amplifier circuit, of which two are presently populated on the wire-wrap area. The circuit is provided with three jumpers to provide flexibility in selecting gain ranges and input offsets, such as are experienced with the proximeter. Diode protection prevents accidental damage to the circuits, should the input voltage become excessive.

Sheet 6, Figure B6: This shows the pulse-width-modulation (PWM) board, which is mounted on the proof-mass actuator. Each end of the coil can be switched independently, so that, with suitable digital program logic, high currents can be avoided with the actuator in a quiescent state. Although there is a 15V supply, the circuit only provides an 8V differential, which provides about one Ampere of current in either direction. Planned improvements

FIGURE B1

Sheet 1: Drivers

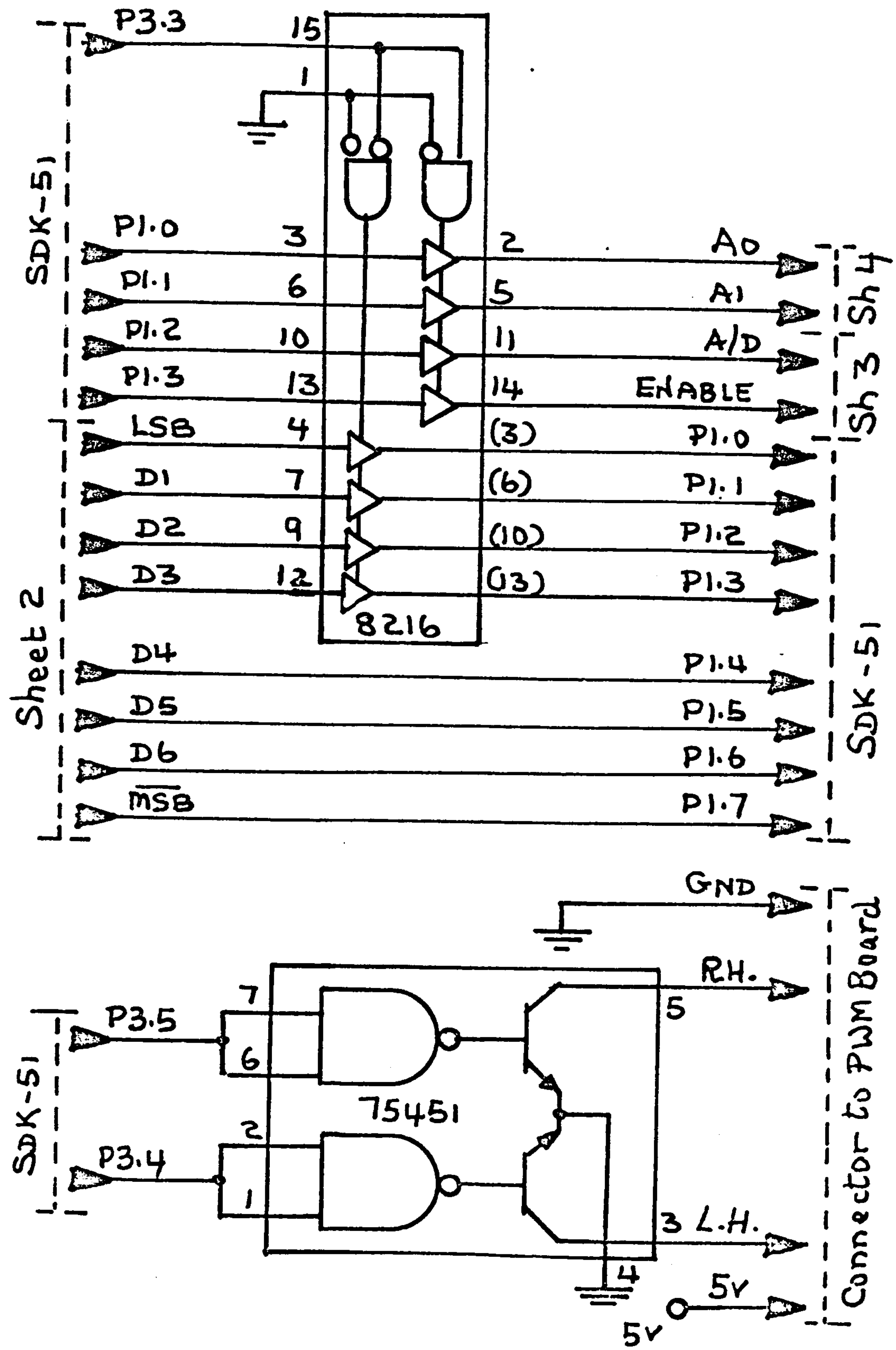


FIGURE B2

Sheet 2: A/D Converter

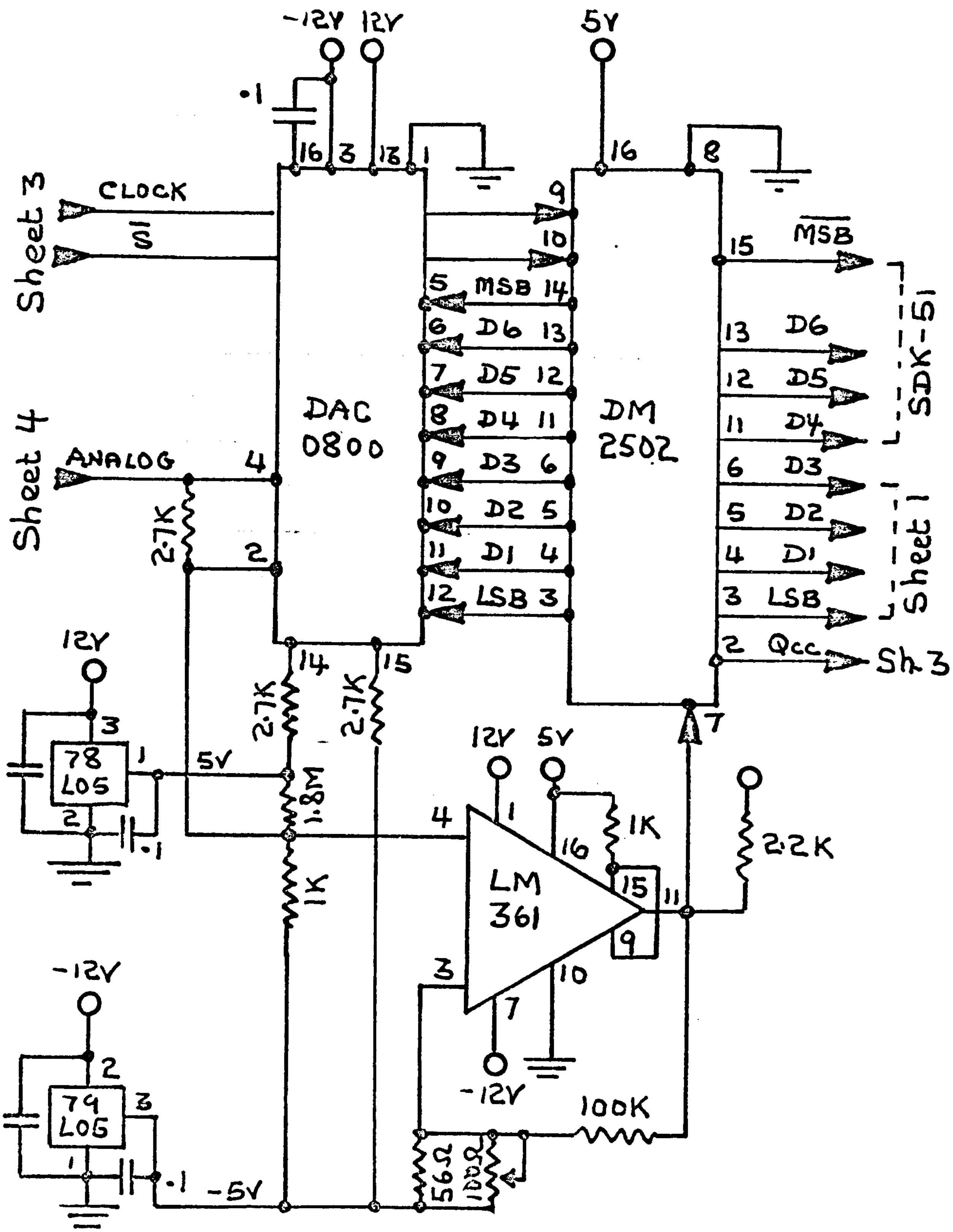


FIGURE B3 .

Sheet 3: A/D Trigger and Clock

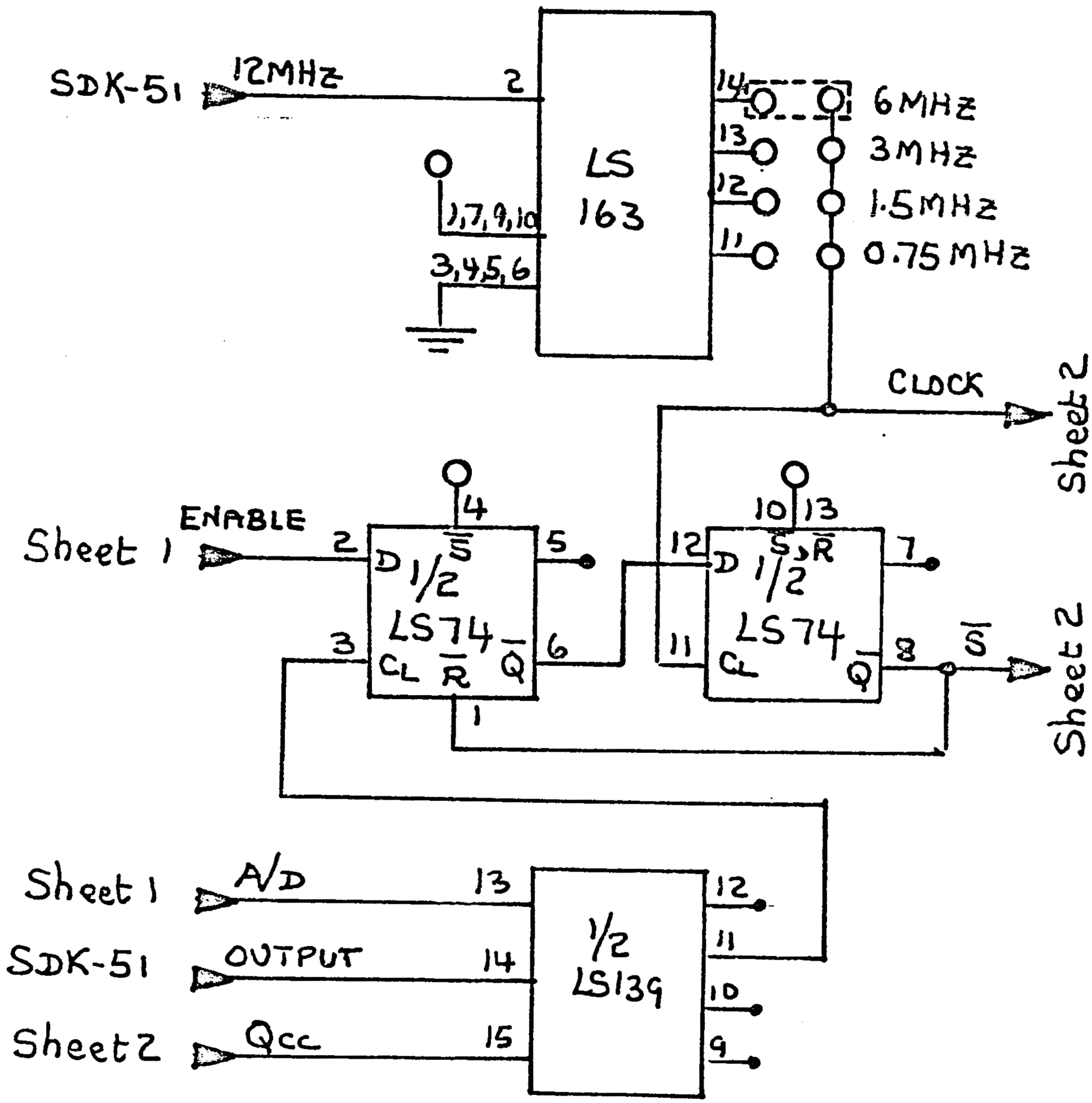


FIGURE B4

Sheet 4: Channel Select

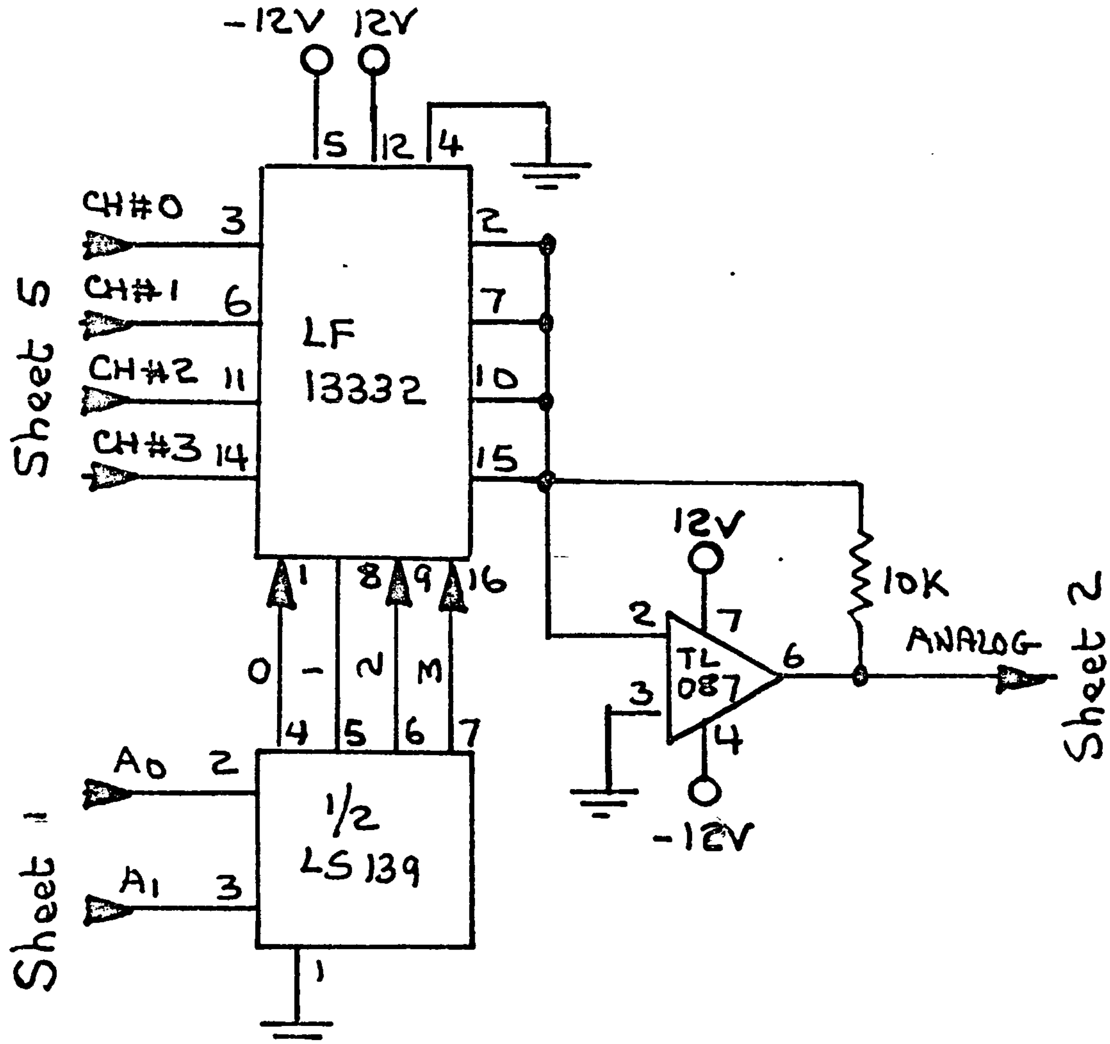


FIGURE B5

Sheet 5: Analog Input Pórt (Typical)

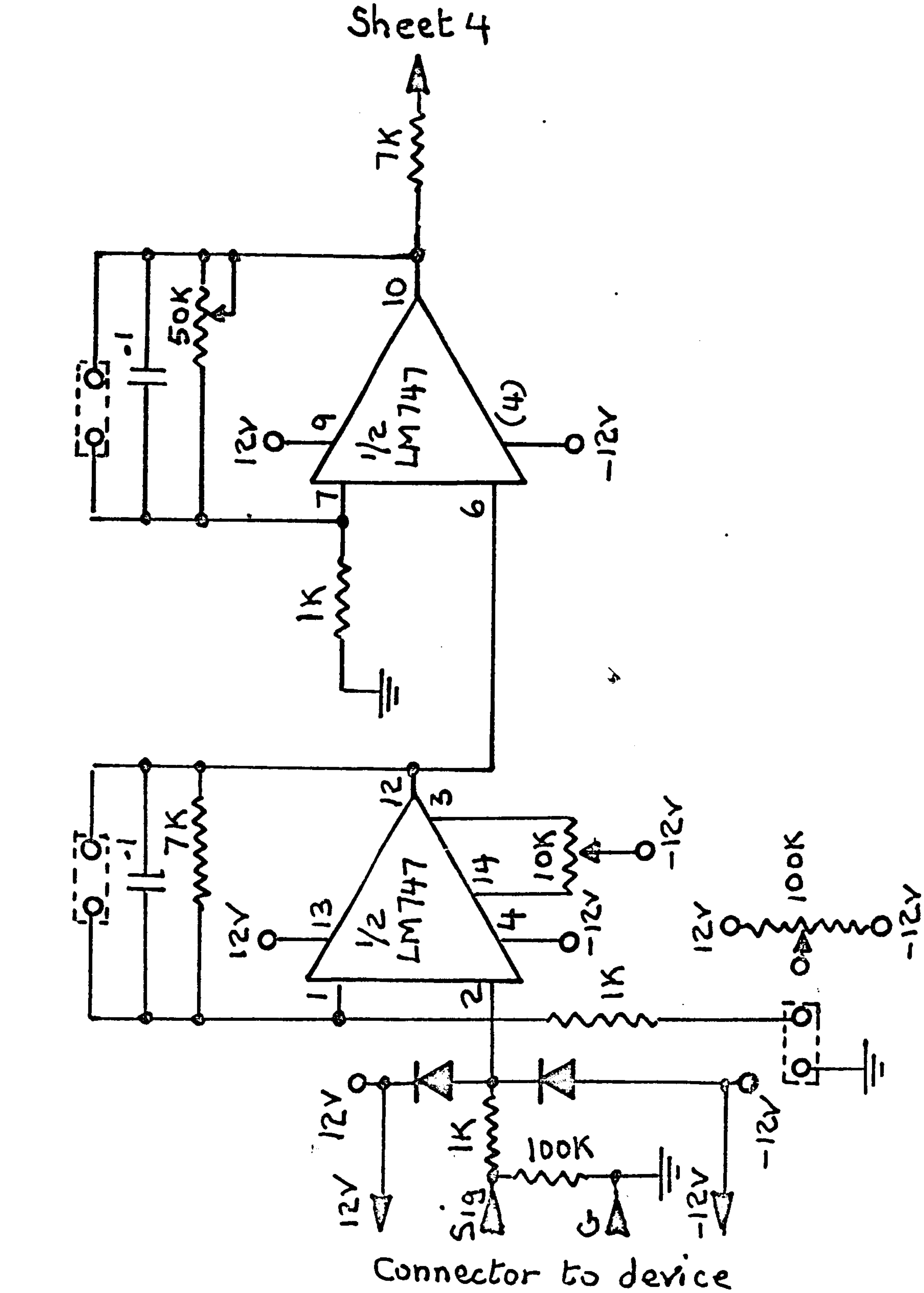
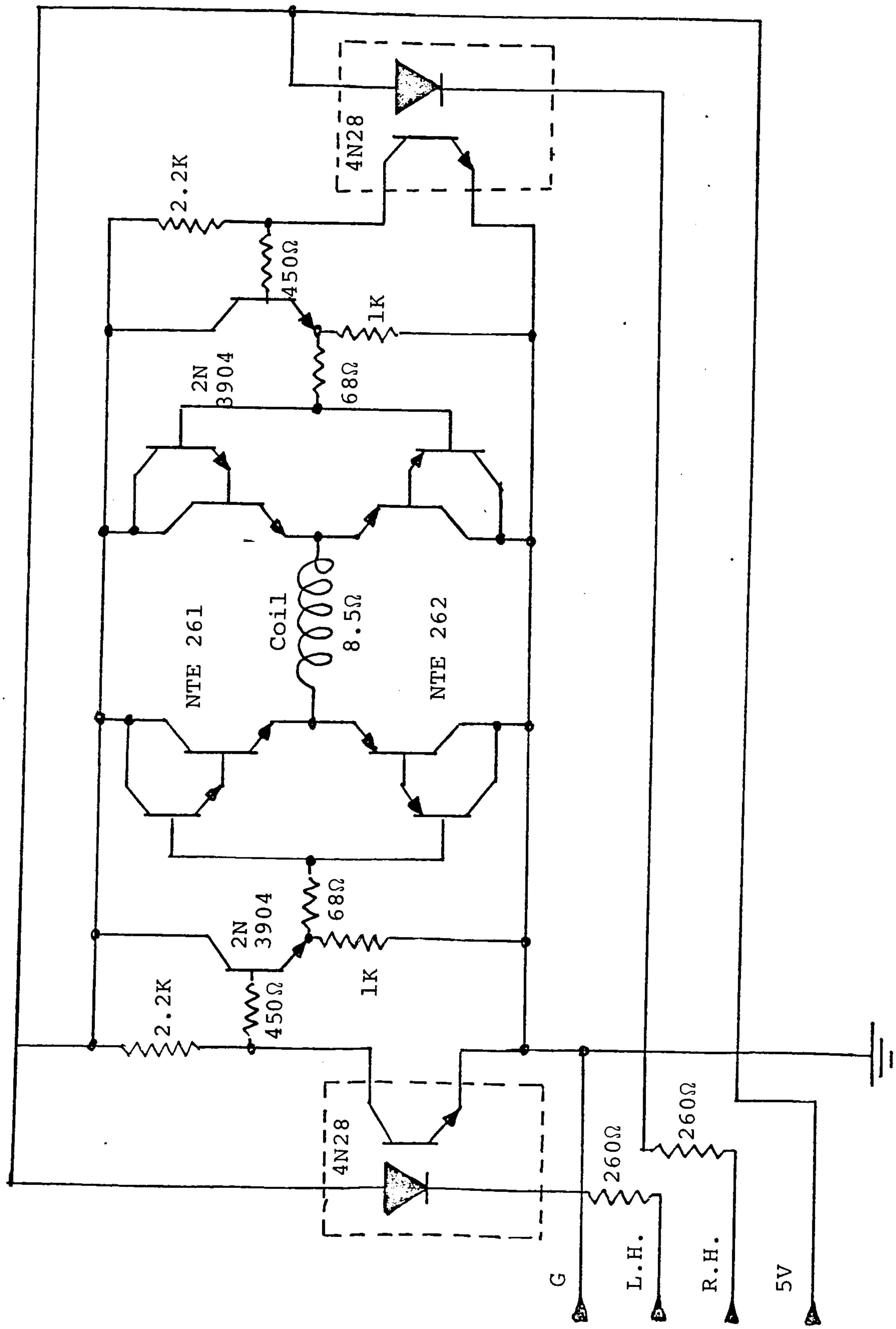


Figure B6

Sheet 6: PWM Board



Connector to Sheet 1

in this circuit should make at least one and a half Amperes possible. Opto-transistors permit electrical isolation of this board, with its own power supply, from the SDK-51 board.